



Share your
information

eZ Publish 3.7
Technical Manual

S y e u r in fo rm at ion

©1999 – 2012 eZ Systems AS

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be downloaded from <http://www.gnu.org/copyleft/fdl.html>.

Corrections and/or suggestions might be sent to info@ez.no.

This PDF file is generated automatically from the online documentation available at <http://doc.ez.no>.

This version was generated on August 3, 2014.

Contents

1	Installation	19
1.1	Normal installation	21
1.1.1	Requirements for doing a normal installation	22
1.1.2	Installing eZ publish on a Linux/UNIX based system	25
1.1.3	Installing eZ publish on Windows	28
1.2	Bundled installation	31
1.2.1	Requirements for doing a bundled installation	32
1.2.2	Installing an eZ publish bundle on a Linux based system	33
1.2.3	Installing an eZ publish bundle on Windows	38
1.3	Manual installation	42
1.3.1	Requirements for doing a manual installation	43
1.3.2	Manual installation on a Linux/UNIX based system	44
1.3.3	Manual installation on Windows	45
1.3.4	Manual configuration of eZ publish	46
1.4	Automated installation	52
1.4.1	Requirements for doing an automated installation	53
1.4.2	Automated installation of eZ publish	54
1.5	The setup wizard	58
1.6	Virtual host setup	70
1.6.1	Virtual host example	73
1.7	Upgrading	76
1.7.1	from 3.5.2 to 3.7.0	77
1.7.2	from 3.7.x to 3.7.y	82
1.8	Removing eZ Publish	85

1.8.1	Removing an eZ Publish bundle	87
1.9	Extensions	89
1.9.1	Extracting the files	90
1.9.2	Activating the extension	92
1.10	Troubleshooting	94
2	Concepts and basics	96
2.1	Designs	97
2.1.1	Design combinations	99
2.2	The internal structure of eZ publish	101
2.2.1	Directory structure	103
2.3	Content and design	105
2.3.1	Storage	107
2.4	Content management	108
2.4.1	Datatypes	110
2.4.2	The content class	111
2.4.3	Class attributes	114
2.4.4	Object versioning	117
2.4.5	Multiple languages	121
2.4.6	The content node	124
2.4.7	The content node tree	127
2.4.8	Top level nodes	130
2.4.9	Node visibility	132
2.4.10	Object relations	135
2.4.11	Sections	136
2.4.12	URL storage	138
2.4.13	Information collection	139
2.5	Configuration	140
2.5.1	Site management	142
2.5.2	Extension siteaccess settings	145
2.5.3	Access methods	146
2.6	Modules and views	149
2.7	URL translation	152

2.8	Access control	156
2.9	Webshop	159
2.10	Workflows	163
3	Templates	165
3.1	Template basics	166
3.1.1	Node templates	169
3.1.2	System templates	171
3.2	The pagelayout	173
3.2.1	The page head	177
3.2.2	Variables in pagelayout	181
3.3	The template language	189
3.3.1	Comments	191
3.3.2	Variable types	192
3.3.3	Variable usage	196
3.3.4	Array and object inspection	201
3.3.5	Control structures	205
3.3.6	Functions and operators	209
3.4	Basic template tasks	211
3.4.1	URL handling	214
3.5	Information extraction	217
3.5.1	Outputting node and object data	219
3.6	The template override system	222
3.6.1	Template override example	224
4	Features	227
4.1	Notifications	228
4.1.1	Using the admin interface	230
4.1.2	Using an actual site	236
4.1.3	Adding a "Keep me updated" button	239
4.1.4	Customizing the E-mails	240
4.1.5	Granting access to notifications	241
4.1.6	Notification events	247

4.1.7	Notification handlers	250
4.1.8	Frequently Asked Questions	253
4.2	Search engine	255
4.3	WebDAV	258
4.3.1	Setting it up	264
5	Reference	267
5.1	Datatypes	268
5.1.1	Authors	270
5.1.2	Checkbox	272
5.1.3	Date	274
5.1.4	Date and time	276
5.1.5	E-mail	278
5.1.6	Enum	279
5.1.7	File	280
5.1.8	Float	284
5.1.9	Identifier	286
5.1.10	Image	288
5.1.11	Ini setting	292
5.1.12	Integer	293
5.1.13	ISBN	295
5.1.14	Keywords	297
5.1.15	Matrix	299
5.1.16	Media	301
5.1.17	Multi-option	304
5.1.18	Object relation	306
5.1.19	Object relations	308
5.1.20	Option	310
5.1.21	Price	312
5.1.22	Range option	314
5.1.23	Selection	316
5.1.24	Subtree subscription	318
5.1.25	Text block	319

5.1.26	Text line	321
5.1.27	Time	323
5.1.28	URL	325
5.1.29	User account	327
5.1.30	XML block	329
5.2	Content classes	344
5.2.1	Content	345
5.2.2	Media	361
5.2.3	Users	368
5.3	Modules	371
5.3.1	class	373
5.3.2	collaboration	392
5.3.3	content	407
5.3.4	error	527
5.3.5	ezinfo	528
5.3.6	form	533
5.3.7	infocollector	536
5.3.8	layout	541
5.3.9	notification	546
5.3.10	package	558
5.3.11	pdf	581
5.3.12	reference	585
5.3.13	role	586
5.3.14	rss	594
5.3.15	search	600
5.3.16	section	603
5.3.17	setup	617
5.3.18	shop	618
5.3.19	trigger	646
5.3.20	url	649
5.3.21	user	658
5.3.22	workflow	683

5.4	Views	695
5.5	Objects	696
5.5.1	ezauthor	699
5.5.2	ezbasket	700
5.5.3	ezbinaryfile	702
5.5.4	ezcontentbrowsebookmark	703
5.5.5	ezcontentbrowserecent	704
5.5.6	ezcontentclass	705
5.5.7	ezcontentclassattribute	708
5.5.8	ezcontentclassclassgroup	710
5.5.9	ezcontentclassgroup	711
5.5.10	ezcontentobject	712
5.5.11	ezcontentobjectattribute	717
5.5.12	ezcontentobjecttranslation	720
5.5.13	ezcontentobjecttreenode	721
5.5.14	ezcontentobjectversion	725
5.5.15	ezdate	728
5.5.16	ezdatetime	729
5.5.17	ezimagealiashandler	730
5.5.18	ezimagedlayer	734
5.5.19	ezimageobject	735
5.5.20	ezinformationcollection	736
5.5.21	ezinformationcollectionattribute	737
5.5.22	ezkeyword	738
5.5.23	ezlocale	739
5.5.24	ezmatrix	742
5.5.25	ezmedia	745
5.5.26	ezmultioption	747
5.5.27	eznodeassignment	748
5.5.28	ezoption	750
5.5.29	ezorder	751
5.5.30	ezorderstatus	754

5.5.31	ezpolicy	755
5.5.32	ezprice	756
5.5.33	ezproductcollectionitem	757
5.5.34	ezrangeoption	758
5.5.35	ezrole	759
5.5.36	ezsection	760
5.5.37	ezsimplifiedxmlinput	761
5.5.38	ezsubtreenotificationrule	762
5.5.39	eztime	763
5.5.40	ezurl	764
5.5.41	ezuser	765
5.5.42	ezvattype	767
5.5.43	ezxhtmlxmloutput	768
5.5.44	ezxmlinputhandler	769
5.5.45	ezxmloutputhandler	770
5.5.46	ezxmltext	771
5.6	Workflow events	772
5.6.1	Approve	773
5.6.2	Multiplexer	775
5.6.3	Payment gateway	776
5.6.4	Simple shipping	778
5.6.5	Wait until date	779
5.7	Template operators	780
5.7.1	Arrays	781
5.7.2	Data and information extraction	805
5.7.3	Formatting and internationalization	815
5.7.4	Images	827
5.7.5	Logical operations	837
5.7.6	Mathematics	865
5.7.7	Miscellaneous	890
5.7.8	Strings	916
5.7.9	URLs	960

5.7.10	Variable and type handling	968
5.8	Template functions	998
5.8.1	Debugging	999
5.8.2	Miscellaneous	1004
5.8.3	Variables	1017
5.8.4	Visualization	1030
5.9	Template control structures	1050
5.9.1	Conditional control	1051
5.9.2	Deprecated	1057
5.9.3	Looping	1059
5.10	Template override conditions	1065
5.10.1	class/edit.tpl	1067
5.10.2	class/groupedit.tpl	1068
5.10.3	class/view.tpl	1069
5.10.4	content/advancedsearch.tpl	1070
5.10.5	content/browse.tpl	1071
5.10.6	content/collectedinfo/*.tpl	1072
5.10.7	content/collectedinfo/*.tpl	1073
5.10.8	content/collectedinfomail/*.tpl	1074
5.10.9	content/datatype/edit/*.tpl	1075
5.10.10	content/datatype/view/*.tpl	1076
5.10.11	content/edit.tpl	1077
5.10.12	content/search.tpl	1078
5.10.13	content/versions.tpl	1079
5.10.14	content/versionview.tpl	1080
5.10.15	layout/set.tpl	1081
5.10.16	node/view/*.tpl	1082
5.10.17	node/view/pdf.tpl	1083
5.10.18	pagelayout.tpl	1084
5.10.19	workflow/edit.tpl	1085
5.10.20	workflow/groupedit.tpl	1086
5.10.21	workflow/view.tpl	1087

5.11	Template fetch functions	1088
5.12	Template PDF functions	1089
5.12.1	anchor	1091
5.12.2	create_index	1092
5.12.3	filled_circle	1093
5.12.4	filled_rectangle	1095
5.12.5	footer	1097
5.12.6	footer_block	1099
5.12.7	frame_header	1100
5.12.8	frontpage	1102
5.12.9	header	1103
5.12.10	header_block	1105
5.12.11	image	1106
5.12.12	keyword	1108
5.12.13	line	1109
5.12.14	link	1111
5.12.15	new_line	1112
5.12.16	new_page	1113
5.12.17	page_number	1114
5.12.18	set_font	1115
5.12.19	set_margin	1117
5.12.20	strike	1119
5.12.21	table	1120
5.12.22	text	1122
5.12.23	text_box	1125
5.12.24	text_frame	1127
5.12.25	toc	1129
5.12.26	ul	1130
5.13	Configuration files	1132
5.13.1	binaryfile.ini	1135
5.13.2	browse.ini	1136
5.13.3	collaboration.ini	1137

5.13.4 collect.ini	1138
5.13.5 content.ini	1139
5.13.6 contentstructuremenu.ini	1161
5.13.7 cronjob.ini	1162
5.13.8 datatype.ini	1171
5.13.9 datetime.ini	1172
5.13.10dbschema.ini	1173
5.13.11debug.ini	1174
5.13.12design.ini	1175
5.13.13error.ini	1184
5.13.14extendedattributefilter.ini	1185
5.13.15ezxml.ini	1186
5.13.16fetchalias.ini	1187
5.13.17file.ini	1188
5.13.18i18n.ini	1189
5.13.19icon.ini	1192
5.13.20image.ini	1193
5.13.21layout.ini	1194
5.13.22ldap.ini	1195
5.13.23logfile.ini	1197
5.13.24menu.ini	1198
5.13.25module.ini	1199
5.13.26notification.ini	1200
5.13.27override.ini	1201
5.13.28package.ini	1202
5.13.29paymentgateways.ini	1203
5.13.30setup.ini	1204
5.13.31shopaccount.ini	1205
5.13.32site.ini	1206
5.13.33soap.ini	1421
5.13.34staticcache.ini	1422
5.13.35template.ini	1429

5.13.36textfile.ini	1430
5.13.37texttoimage.ini	1431
5.13.38toolbar.ini	1432
5.13.39transform.ini	1433
5.13.40units.ini	1434
5.13.41upload.ini	1435
5.13.42viewcache.ini	1436
5.13.43webdav.ini	1437
5.13.44wordtoimage.ini	1438
5.13.45workflow.ini	1439
5.14 Libraries	1449
5.14.1 ezdb	1450
5.14.2 ezdbschema	1451
5.14.3 ezfile	1452
5.14.4 ezi18n	1453
5.14.5 ezimage	1454
5.14.6 ezlocale	1455
5.14.7 ezpdf	1456
5.14.8 ezsoap	1457
5.14.9 eztemplate	1458
5.14.10ezutils	1459
5.14.11ezwebdav	1460
5.14.12ezxml	1461
5.15 XML tags	1462

List of Figures

1.1	Step 1: Main menu	34
1.2	Step 2: Confirmation	34
1.3	Step 3: File extraction	35
1.4	Step 4: Network interface	36
1.5	Step 5: Web server port	36
1.6	Step 6: Summary	37
1.7	Step 1: Welcome dialog	39
1.8	Step 2: Component selection	39
1.9	Step 3: Destination folder	40
1.10	Step 4: Start menu	40
1.11	Step 5: Installation in progress	41
1.12	Step 6: Completion	41
1.13	Step 1: Welcome page	59
1.14	Step 2: Issues	60
1.15	Step 3: Outgoing E-mail	61
1.16	Step 4: Database choice	62
1.17	Step 5: Database initialization	62
1.18	Step 6: Language support	63
1.19	Step 7: Site type	64
1.20	Step 8: Site functionality	64
1.21	Step 9: Site access configuration	65
1.22	Step 10: Site details	66
1.23	Step 11: Site administrator	67
1.24	Step 12: Site registration	68

1.25	Step 13: Finished	69
1.26	Screenshot of extension configuration in administration interface.	92
1.27	The debug output appears at the bottom of the page	95
2.1	The design fallback mechanism.	99
2.2	Libraries, kernel and modules.	101
2.3	Content + Design = Web page	106
2.4	Storage overview	107
2.5	Example of a content class.	111
2.6	The class edit interface.	112
2.7	Example of a content object that consists of two versions.	117
2.8	Overview of the object states.	120
2.9	Content object structure (with versions and translations).	121
2.10	Object - node relation	124
2.11	Objects, nodes and the content node tree	127
2.12	Content node tree	128
2.13	Objects, node and the content node tree - multiple locations	128
2.14	Content node tree with multiple locations	129
2.15	Top level nodes	130
2.16	Hiding a visible node	133
2.17	Hiding an invisible node	133
2.18	Unhiding a node with a visible ancestor	134
2.19	Unhiding a node with an invisible ancestor	134
2.20	Example of sections.	137
2.21	Example of a setup with two siteaccesses.	142
2.22	Siteaccess directory example.	143
2.23	Configuration override example.	144
2.24	Objects, nodes and the URL table.	154
2.25	Users, groups, policies and roles.	156
2.26	The integrated e-commerce solution.	159
2.27	The workflow system.	163
3.1	Client - server cycle.	167

3.2	The module result as a part of the pagelayout.	167
3.3	Location of pagelayout and full view template in example design.	169
3.4	Pagelayout + node view full template.	170
3.5	The location of the pagelayout (main) template.	173
3.6	The structure of the "ezdate" object.	195
3.7	Typical components of a function call.	209
3.8	Typical components of a template operator call.	210
3.9	The override system.	222
3.10	Template override example.	223
3.11	Example content node tree.	224
3.12	Pagelayout + override templates in example design.	225
3.13	Template override example.	226
4.1	The notification filter interface.	229
4.2	Browsing the content tree.	231
4.3	Subscribing to subtree notifications using the context menu.	232
4.4	The "notification added" confirmation for administrators.	232
4.5	Notification settings for administrators.	233
4.6	Browsing the content tree.	234
4.7	The "Up" button	234
4.8	Digest settings	234
4.9	The list of items for subtree notifications.	235
4.10	Settings for collaboration notifications.	235
4.11	The "keep me updated" button.	236
4.12	The "notification added" confirmation for users.	236
4.13	Notification settings for users.	237
4.14	The usergroup view interface.	241
4.15	The list of roles.	242
4.16	Adding a new role.	242
4.17	The new policy wizard, step 1.	243
4.18	The new policy wizard, step 2.	244
4.19	The role edit interface.	244
4.20	The role view interface.	245

4.21	Assigning a role to a user group.	245
4.22	The role view interface.	246
4.23	Standard search interface	255
4.24	Advanced search interface	256
4.25	Search statistics	257
4.26	WebDAV - Virtual top folder	258
4.27	WebDAV - Login	259
4.28	WebDAV - Top level nodes	259
4.29	WebDAV - Content node tree	260
4.30	WebDAV - IE open dialog	265
4.31	WebDAV - Content node tree	266
5.1	Class attribute edit interface for the "Authors" datatype.	270
5.2	Object attribute edit interface for the "Authors" datatype.	271
5.3	Class attribute edit interface for the "Checkbox" datatype.	272
5.4	Object attribute edit interface for the "Checkbox" datatype.	272
5.5	Class attribute edit interface for the "Date" datatype.	274
5.6	Object attribute edit interface for the "Date" datatype.	275
5.7	Class attribute edit interface for the "Datetime" datatype.	276
5.8	Object attribute edit interface for the "Date and time" datatype.	277
5.9	Class attribute edit interface for the "Email" datatype.	278
5.10	Object attribute edit interface for the "E-mail" datatype.	278
5.11	Class attribute edit interface for the "File" datatype.	280
5.12	Object attribute edit interface for the "File" datatype.	281
5.13	Object attribute edit interface for the "File" datatype.	281
5.14	Complete directory structure with uploaded files.	282
5.15	Class edit interface for the "Float" datatype.	284
5.16	Object attribute edit interface for the "Float" datatype.	285
5.17	Class attribute edit interface for the "Identifier" datatype.	287
5.18	Class attribute edit interface for the "Image" datatype.	288
5.19	Object attribute edit interface for the "Image" datatype.	289
5.20	Object attribute edit interface for the "Image" datatype.	289
5.21	Example of image path on the filesystem.	290

5.22	Example of an image subdirectory	290
5.23	Complete directory structure with uploaded image and generated variations.	291
5.24	Class edit interface for the "Integer" datatype.	293
5.25	Object attribute edit interface for the "Integer" datatype.	294
5.26	Class attribute edit interface for the "ISBN" datatype.	295
5.27	Object attribute interface for the "ISBN" datatype.	295
5.28	Class attribute edit interface for the "Keywords" datatype.	297
5.29	Object attribute edit interface for the "Keywords" datatype.	298
5.30	Class attribute edit interface for the "Matrix" datatype.	299
5.31	Object attribute edit interface for the "Matrix" datatype.	300
5.32	Class attribute edit interface for the "Media" datatype.	301
5.33	Object attribute edit interface for the "Media" datatype (Flash).	302
5.34	Object attribute edit interface for the "Media" datatype (QuickTime).	302
5.35	Object attribute edit interface for the "Media" datatype (Real Media).	303
5.36	Object attribute edit interface for the "Media" datatype (Windows media).	303
5.37	Class attribute edit interface for the "Multi-option" datatype.	304
5.38	Object attribute edit interface for the "Multi-option" datatype.	305
5.39	Class attribute edit interface for the "Object relation" datatype.	306
5.40	Object attribute edit interface for the "Object relation" datatype.	307
5.41	Class attribute edit interface for the "Object relations" datatype.	308
5.42	Object attribute edit interface for the "Object relations" datatype.	309
5.43	Class attribute edit interface for the "Option" datatype.	310
5.44	Object attribute edit interface for the "Option" datatype.	311
5.45	Class attribute edit interface for the "Price" datatype.	312
5.46	Object attribute edit interface for the "Price" datatype.	313
5.47	Class attribute edit interface for the "Range option" datatype.	314
5.48	Object attribute edit interface for the "Range option" datatype.	315
5.49	Class attribute edit interface for the "Selection" datatype.	316
5.50	Object attribute interface for the "Selection" datatype.	317
5.51	Class edit interface for the "Text block" datatype.	319
5.52	Object attribute edit interface for the "Text block" datatype.	320
5.53	Class edit interface for the "Text line" datatype.	321

5.54	Object attribute interface for the "Text line" datatype.	322
5.55	Class attribute edit interface for the "Time" datatype.	323
5.56	Object attribute edit interface for the "Time" datatype.	323
5.57	Class attribute edit interface for the "URL" datatype.	325
5.58	Object attribute edit interface for the "URL" datatype.	325
5.59	Class attribute edit interface for the "User account" datatype.	327
5.60	Object attribute edit interface for the "User account" datatype.	328
5.61	Settings interface for the "User account" datatype.	328
5.62	Class attribute edit interface for the "XML block" datatype.	330
5.63	Object attribute edit interface for the "XML block" datatype.	330
5.64	Edit interface for the "Approve" event.	773
5.65	Edit interface for the "Multiplexer" event.	775
5.66	Edit interface for the "Payment gateway" event.	776
5.67	Edit interface for the "Simple shipping" event.	778
5.68	Edit interface for the "Wait until date" event.	779
5.69	Text rendered as image using the 1942 font.	834
5.70	Text rendered as image using the a_d_mono font.	834
5.71	Text rendered as image using the archtura font.	834
5.72	Text rendered as image using the arial font.	835
5.73	Text rendered as image using the gallery font.	835
5.74	Text rendered as image using the object_text font.	835
5.75	Text rendered as image using the sketchy font.	836
5.76	Text rendered as image using the smartie font.	836
5.77	Text rendered as image using the a_d_mono font.	836
5.78	The content tree	911

Chapter 1

Installation

This chapter explains how to obtain and install eZ Publish using the different installation methods. In addition, it also describes how to upgrade or remove an existing eZ Publish installation. If you don't want to install eZ Publish yourself, you can always hire eZ Systems to install and setup the software for you. It is also possible to purchase a hosted eZ Publish solution from various providers and partners.

There are four ways of installing eZ Publish:

1. Normal installation
2. Bundled installation
3. Manual installation
4. Automated installation

Normal installation

This option is the most common and recommended way of installing eZ Publish. It requires a system which already has the proper environment installed, most notably a web server and a database. eZ Publish needs to be downloaded and unpacked. A web-based setup wizard is initiated using a browser. The setup wizard asks a couple of questions and automatically configures eZ Publish. The method is explained under the "Normal installation" (page 21) section.

Bundled installation

A bundled installation contains eZ Publish and additional software that is required to run the system (Apache, PHP, MySQL, ImageMagick, etc.). Almost everything that is included in the package is automatically set up by the installer program. The bundled software is pre-compiled for the Intel x86 hardware architecture. This option is for quick and dirty testing/demonstration purposes only. In other words, it is not designed for production/live environments. This installation method is explained in the "Bundled installation" (page 31) section.

Manual installation

This option is for experienced users. No wizards or fancy dialogs, no bundled software, no installers, no nothing. This method requires a system which already has a web-server and a database set up and ready to go; eZ Publish needs to be downloaded and unpacked. The system is then configured by manually altering various configuration files and making manual changes to the database. This method is explained under the "Manual installation" (page [42](#)) section.

Automated installation

This installation method (also named kickstart) is for experienced users. It is designed for system administrators who wish to do pre-configured installations of eZ Publish that require a minimum of interaction with the web based setup wizard. It requires a system which already has the proper environment installed, most notably a web server and a database. eZ Publish needs to be downloaded and unpacked. Instead of clicking through the setup wizard and manually providing configuration parameters, the system is installed based on a group of settings defined in a configuration file. This method is explained under the "Automated installation" (page [52](#)) section.

1.1 Normal installation

The normal installation method is the most common and recommended way of deploying eZ publish. It requires a system which already has the proper environment installed, most notably a web server and a database. The necessary requirements are explained in detail within the next section. A typical normal installation process consists of the following steps:

- Setting up / creating a database
- Downloading a packaged eZ publish distribution
- Unpacking the eZ publish distribution
- Initiating and going through the web based setup wizard

Once the web based setup wizard has completed, eZ publish will be ready for use.

1.1.1 Requirements for doing a normal installation

eZ publish makes use of and depends on four important things:

1. A web server
2. A server-side PHP scripting engine
3. A database server
4. An image conversion system (optional)

The first three things should be in place before an eZ publish installation is deployed. The image conversion system is optional and is only needed if you're planning to use eZ publish with images. The web server and the server-side PHP scripting engine has to run on the same machine. The database server may run on a different computer. For the moment, the following software solutions can be used:

Web server

Currently, only the Apache web server is supported. It is recommended to use the latest version of the 1.3 branch. However, it is possible to use the 2.x series. When using Apache 2.x, it must run in "prefork" mode instead of "threaded" mode - the reason for this is because the PHP libraries are not threadsafe. Please note that Apache 2.x for Windows only exists in "threaded" mode and thus it should not be used to run an eZ publish solution on Windows. The Apache web server is the most popular web server on the planet. It is free, open source and can be downloaded from <http://www.apache.org>.

Server-side PHP scripting engine

Since most of the eZ publish system is written using the PHP scripting language, a PHP (hypertext preprocessor) server-side engine is needed. Make sure you have PHP 4.4. It is recommended to use the latest version of the 4.4 branch. Please note that eZ publish will not work correctly with PHP 5 and thus PHP 5 should not be used. PHP needs to have compiled-in support for either MySQL or PostgreSQL.

PHP is free software and can be downloaded from <http://www.php.net>.

PHP CLI

It is strongly recommended to have [PHP CLI](#) installed, otherwise some features will not be available.

PHP memory limit issue

eZ publish needs at least 64 MB in order to complete the setup wizard. This means that you'll have to increase the default "memory_limit" setting which is located in the "php.ini" configuration file. (Don't forget to restart apache after editing "php.ini".) Normal operation requires about 16 MB. However, it is highly recommended that you keep the 64 MB setting since eZ publish consumes a lot more memory as soon as you use PDF export feature, reindex the search, etc. Multi-lingual sites that store the content in Unicode (UTF-8) will also require at least 64 MB.

Database server

eZ publish stores miscellaneous data structures and actual content using a database. This means that a database server has to be available for eZ publish at all times. By default, eZ publish is compatible with the following database solutions:

- MySQL 3.23 or later (<http://www.mysql.com>)
Note that MySQL 5.x is currently not supported.
- PostgreSQL (<http://www.postgresql.org>)

The setup wizard will automatically detect the database server as long as it is running on the same computer that functions as the web server.

If you are going to use PostgreSQL, make sure the "pgcrypto" module is installed. On Linux/UNIX, you may need to install a separate package called "postgresql-contrib" (refer to the [PostgreSQL documentation](#) for more information), which contains the "pgcrypto" module. The "pgcrypto" module provides cryptographic functions for PostgreSQL, including the "digest" function, which is needed for eZ Publish. When setting up a PostgreSQL database for eZ Publish, you will have to register these functions in the database as described in the "Setting up a database" part of the installation instructions.

Oracle compatibility

The eZ Publish Extension for Oracle Database makes it possible to use Oracle as a database for eZ Publish. Please refer to the documentation of the database extension for technical details.

Image conversion system (optional)

In order to scale, convert or modify images, eZ publish needs to make use of an image conversion system. One of the following software packages (both are free) can be used:

- GD (comes with PHP)
- ImageMagick (<http://www.imagemagick.org>)

ImageMagick supports more formats than GD and usually produces better results (better scaling, etc.). The setup wizard will automatically detect the pre-installed image conversion system(s).

The installation and setup of required software solutions (outlined above) is far beyond the scope of this document. Please refer to the homepage and documentation of the different software solutions.

your
information

1.1.2 Installing eZ publish on a Linux/UNIX based system

The requirements for doing a normal installation must be met. Please read the previous section if you're not sure about the requirements. Proceed only if you have access to a UNIX based environment with Apache, PHP, MySQL or PostgreSQL already installed and running. (Please note that PHP 4.4, PHP 5 should not be used.) As mentioned earlier, the database server may run on a different computer than the web server. This section will guide you through the following steps:

- Setting up a database (MySQL or PostgreSQL)
- Downloading eZ publish
- Unpacking eZ publish
- Initiating the setup wizard

Setting up a database

A database must be created before the initiation of the setup wizard takes place. The following text explains how to set up a database using either MySQL or PostgreSQL.

MySQL

1. Login as root:

```
$ mysql -u <mysql_username> -p<mysql_password>
```

The MySQL client should display a "mysql>" prompt.

2. Create a new database:

```
mysql> create database <database> character set <character_set>;
```

3. Grant access permissions:

```
mysql> grant all on <database>.* to <username>@<host> identified by  
'<password>';
```

mysql_username	The MySQL user (if no user is set up, use "root").
mysql_password	The password that belongs to the mysql_username.
username	The username that will be used to access the

	database.
password	The password you wish to set in order to limit access to the database.
database	The name of the database, for example "my_new_database".
host	The hostname of the server on which eZ publish will be running. (may be "localhost" if MySQL is installed on the same server).
character_set	The character encoding scheme to be used in the database.

PostgreSQL

1. Become the PostgreSQL super user (normally called postgres):

```
$ su <postgres_super_user>
```

2. Create a PostgreSQL user:

```
$ createuser <username>
```

3. Create a database:

```
$ createdb -E <encoding> <database>
```

4. Import the "pgcrypto" module into the database:

```
$ psql <database> < /usr/share/pgsql/contrib/pgcrypto.sql
```

username	The username that will be used to access the database.
database	The name of the database, for example "my_new_database".
encoding	The character encoding scheme to be used in the database.

Downloading eZ publish

The latest stable version of eZ publish can be downloaded from http://ez.no/download/ez_publish.

Unpacking eZ publish

Use your favorite tool to unpack the downloaded eZ publish distribution to a web-served directory (a directory that is reachable using a web browser). The following example shows how to do this using the tar utility (to unpack a tar.gz file, assuming that the "tar" and the "gzip" utilities are installed on the system):

```
$ tar zxvf ezpublish-<version_number>.tar.gz -C <web_served_directory>
```

version_number	The version number of eZ publish that was downloaded.
web_served_directory	Full path to a directory that is served by the web server. This can be the path to the document root of the web server, or a personal web-directory (usually called "public_html" or "www", and located inside a user's home directory).

The extraction utility will unpack eZ publish into a subdirectory called "ezpublish-3.x.x". Feel free to rename this directory to something more meaningful, for example "my_site".

Initiating the setup wizard

The setup wizard can be started using a web browser immediately after the previous steps (described in this section) are completed. It will be automatically run the first time someone tries to access/browse the index.php file located in the eZ publish directory. Let's assume that we are using a server with the hostname "www.example.com" and that after unpacking, the eZ publish directory was renamed to "my_site".

Document root example

If eZ publish was unpacked into a directory called "my_site" under the document root, the setup wizard can be initiated by browsing the following URL: http://www.example.com/my_site/index.php.

Home directory example

If eZ publish was unpacked to a web-served directory located inside the home directory of a user with the username "peter", (usually called "public_html", "www", "http", "html" or "web"), the setup wizard can be initiated by browsing the following URL: http://www.example.com/~peter/my_site/index.php.

Please refer to "The setup wizard (page 58)" section for a detailed description of the web based setup wizard.

1.1.3 Installing eZ publish on Windows

The requirements for doing a normal installation must be met! Please read the "Requirements for doing a normal installation" (page 22) section first. Proceed only if you have access to a Windows based system with Apache, PHP, MySQL already installed and running. (Do not use Apache 2.x for Windows and PHP 5.x. Please note that eZ publish is supposed to work with PostgreSQL as well although this combination is not tested on Windows.) As mentioned earlier, the database server may run on a different computer than the web server. This section will guide you through the following steps:

- Setting up a MySQL database
- Downloading eZ publish
- Unpacking eZ publish
- Initiating the setup wizard

Setting up a MySQL database

A database must be created before the initiation of the setup wizard takes place. The following text explains how to set up a database using MySQL:

MySQL

1. Login as root:

```
$ mysql -u <mysql_username> -p<mysql_password>
```

The MySQL client should display a "mysql>" prompt.

2. Create a new database:

```
mysql> create database <database> character set <character_set>;
```

3. Grant access permissions:

```
mysql> grant all on <database>.* to <username>@<host> identified by  
'<password>';
```

mysql_username	The MySQL user (if no user is set up, use "root").
mysql_password	The password that belongs to the mysql_username.
username	The username that will be used to access the

	database.
password	The password you wish to set in order to limit access to the database.
database	The name of the database, for example "my_new_database".
host	The hostname of the server on which eZ publish will be running. (may be 'localhost' if MySQL is installed on the same server).
character_set	The character encoding scheme to be used in the database.

Downloading eZ publish

The latest stable version of eZ publish can be downloaded from http://ez.no/download/ez_publish. Windows users should download the ".zip" archive.

Unpacking eZ publish

Use your favorite utility to unpack the downloaded eZ publish archive to a web-served directory (a directory that is reachable using a web browser). The extraction utility will unpack eZ publish into a subdirectory called "ezpublish-3.x.x". Feel free to rename this directory to something more meaningful, for example "my_site".

Initiating the setup wizard

The setup wizard can be started using a web browser immediately after the previous steps (described in this section) are completed. It will be automatically run the first time someone tries to access/browse the index.php file located in the eZ publish directory. Let's assume that we are using a server with the hostname "www.example.com" and that after unpacking, the eZ publish directory was renamed to "my_site".

Document root example

If eZ publish was unpacked into a directory called "my_site" under the document root, the setup wizard can be initiated by browsing the following URL: http://www.example.com/my_site/index.php.

Home directory example

If eZ publish was unpacked to a web-served directory located inside the home directory of a user with the username "peter", (usually called "public_html", "www", "http", "html" or "web"), the setup wizard can be initiated by browsing the following URL: http://www.example.com/~peter/my_site/index.php.

Please refer to "The setup wizard" (page 58) section for a detailed description of the web based setup wizard.

1.2 Bundled installation

A bundled installation contains everything that is needed for eZ publish to run and eZ publish itself. A bundle contains the following software:

- Apache (web server)
- PHP (server side scripting engine)
- MySQL (database server and client)
- ImageMagick (image conversion software)
- eZ publish

All software included in a bundle is pre-compiled for the Intel x86 hardware architecture. The binaries/executables are dynamically linked to the shared libraries of the target operating system (Linux or Windows). Each bundle comes with an easy-to-use installer. The installer automatically sets up the additional software, which means that no manual configuration of Apache, PHP, MySQL and ImageMagick is needed. The contents of a bundle is put inside a temporary directory, totally isolated from the rest of the filesystem. The software included in the bundle can coexist with previously installed versions of Apache, PHP, MySQL, etc. Bundles are very easy to install and can be just as easily removed. Bundled installations are available for the following operating systems:

- Linux distributions (Debian/Gentoo/Mandrake/RedHat/SuSE/etc.)
- Microsoft Windows 95/98/ME/NT/2000/XP

The bundles are designed for getting eZ publish up and running in the fastest possible way on a computer that doesn't have the required software installed. However, bundles are meant to be used for testing/demonstration purposes only. If you plan to use eZ publish for more than just testing, we recommend that you install it using the normal installation method, which is described under the Normal installation section. The next sections describe the necessary requirements, how to install, use and remove eZ publish bundles.

1.2.1 Requirements for doing a bundled installation

Linux requirements

Root access

An eZ publish bundle can only be installed by the root user. This means that you'll need to have root access on the computer that you wish to install the bundle.

Diskspace

Because an eZ publish bundle contains a lot of additional software (Apache, PHP, MySQL, etc.), it takes up quite a lot of space. You need to have at least 32 MB of free space on the root partition.

Bash

The installer must be executed from within an instance of the "Bourne Again Shell" (Bash).

Windows requirements

Administrator access on NT/2000/XP

When installing on Windows NT, 2000 or XP, you need to be logged in with a user that has administrator privileges (preferably the "Administrator" user).

Diskspace

Because an eZ publish bundle contains a lot of additional software (Apache, PHP, MySQL, etc.), it takes up quite a lot of space. You need to have at least 32 MB of free space on the C: drive.

1.2.2 Installing an eZ publish bundle on a Linux based system

This section describes how to deploy an eZ publish bundle on a Linux based system:

1. Download the latest Linux compatible eZ publish bundle from http://ez.no/download/ez_publish. Look for "ezpublish-x.y.z-Linux-STABLE.i386.tar.gz" where "x.y.z" is the version number, for example "3.7.0".
2. Unpack the bundle:

```
$ tar zxvf ezpublish-x.y.z-Linux-STABLE.i386.tar.gz
```

3. Become the root user:

```
$ su -
```

4. Start the installation script:

```
# cd ezpublish-x.y.z-Linux-STABLE.i386  
# ./install.sh
```

The following text explains the various dialog screens of the installation script.

Main menu

(see figure 1.1)

Select the "Install eZ publish 3" option and hit enter.

Previous installations

The installation script will automatically detect a previously installed bundle by checking a lock file. A previous installation must be removed. Use the "Uninstall eZ publish 3" option from the main menu, the installer will automatically remove the previous bundle. If a previous version was removed manually then the lock file may still be present. Remove the lock file manually by deleting the "/var/state/ezpublish/ezpublish.lock" file.

Confirmation

(see figure 1.2)

A dialog will ask for a final confirmation. Choose "Yes". The installation script will then attempt to install all the software included in the bundle. Everything will be put in the "/opt/ezpublish" directory (except the lock file, which will be located at "/var/state/ezpublish/").

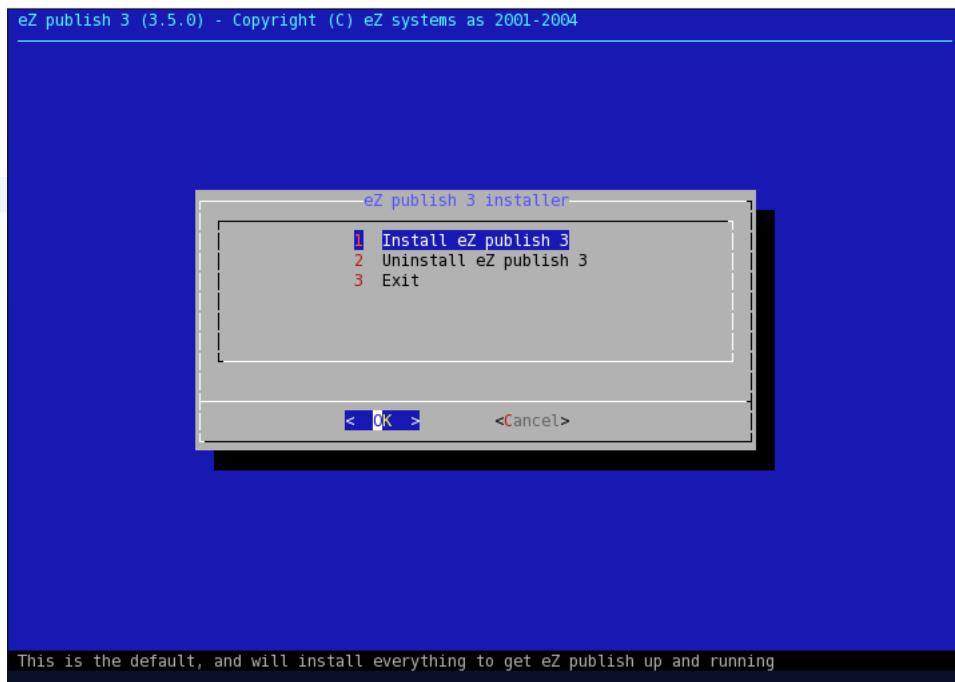


Figure 1.1: Step 1: Main menu

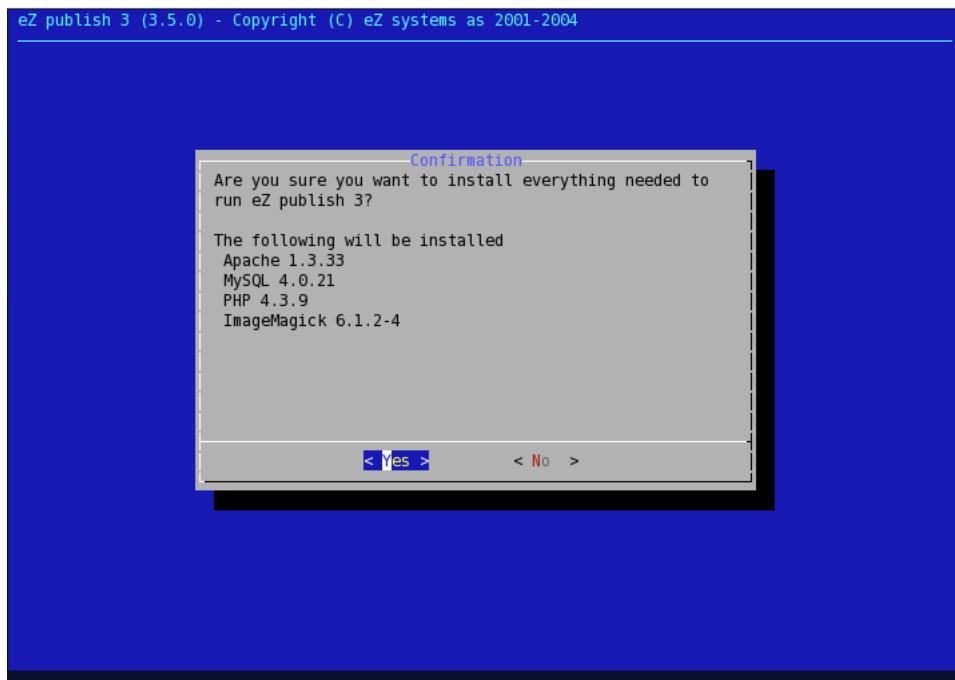


Figure 1.2: Step 2: Confirmation

File extraction

(see figure 1.3)

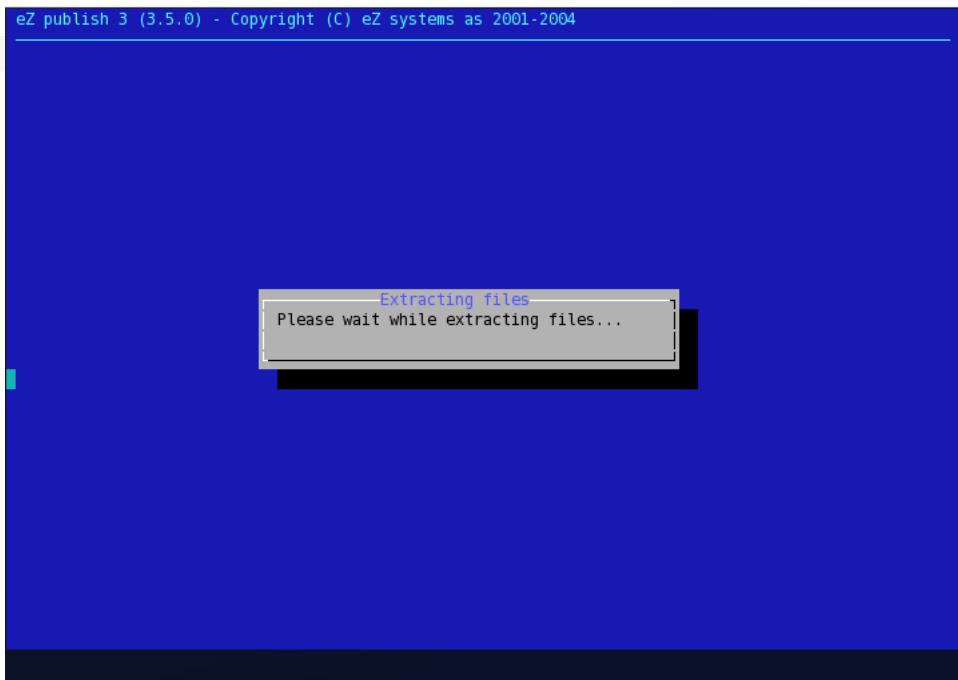


Figure 1.3: Step 3: File extraction

The installation script will extract the files included in the bundle. This may take some time (a minute or two).

Network interface

(see figure 1.4)

A dialog will ask for the name of the primary network interface (most likely the Ethernet controller). The primary network interface is usually called "eth0". In case of doubt: bring up a shell and use the "ifconfig" command to figure out the name of the interface. It is possible to use the local loopback interface if no physical network interface is present. The name of the local loopback interface is "lo".

Web server port

(see figure 1.5)

The installation script will ask for a TCP port number. The bundled web server (Apache) will communicate with the rest of the world through this port. The port must not be used by any other service running on the system. The default port for HTTP traffic is 80. If a web server is already using this port, then another port must be chosen (for example 8085).

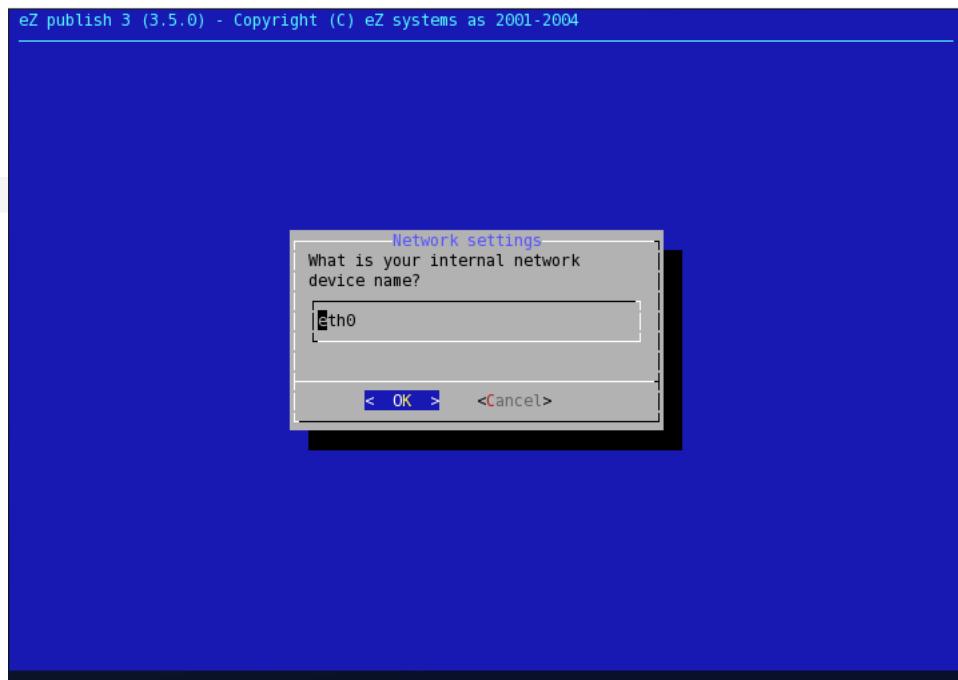


Figure 1.4: Step 4: Network interface

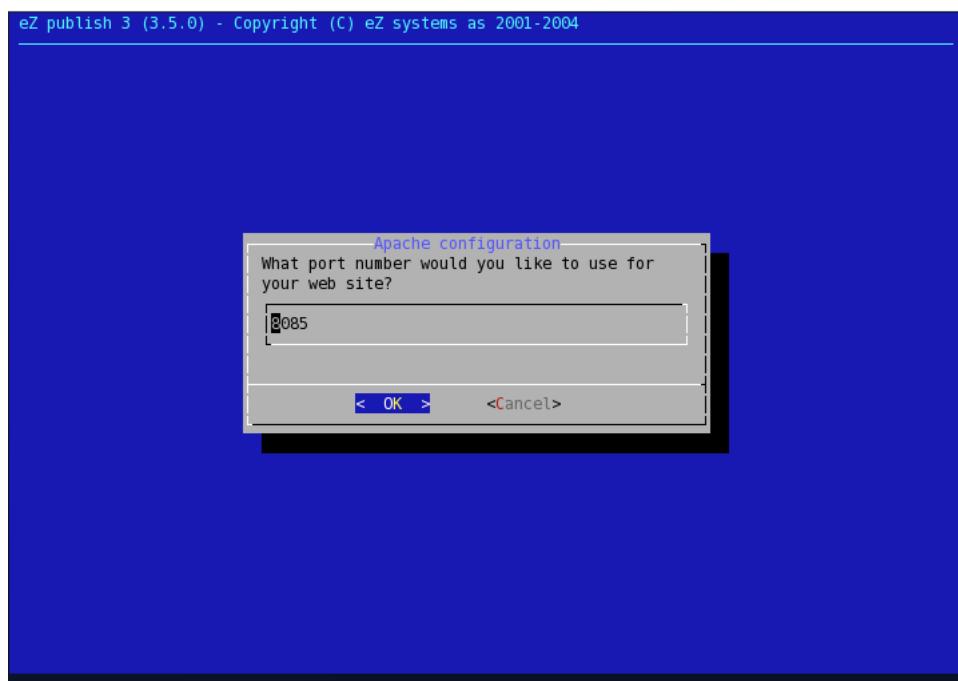


Figure 1.5: Step 5: Web server port

Finish

(see figure 1.6)

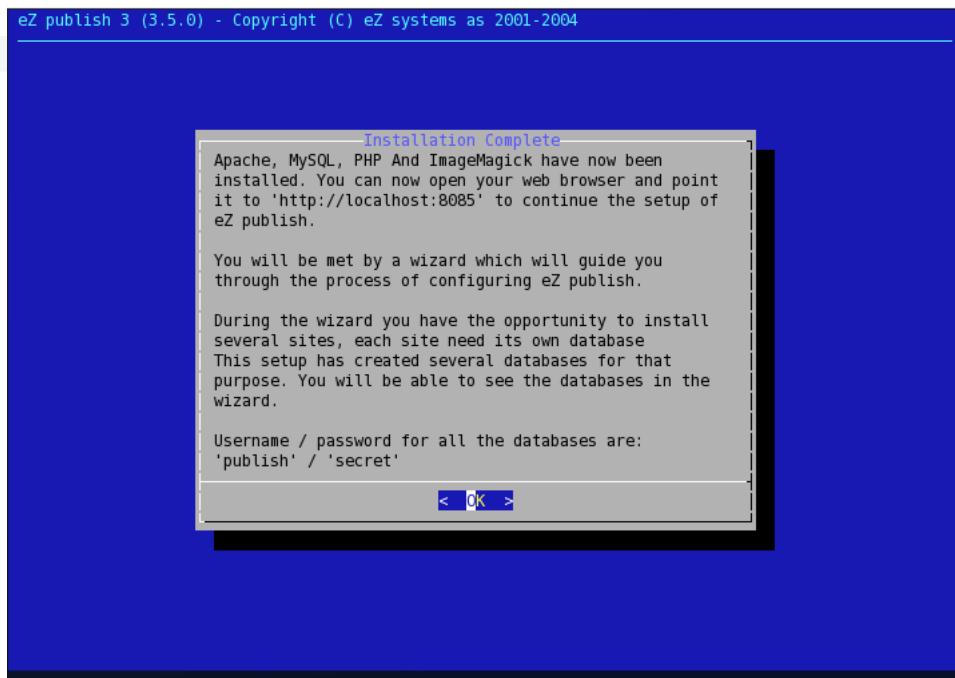


Figure 1.6: Step 6: Summary

The final dialog will display information about the installation and how the eZ publish system can be reached using a web browser. The address will be something like "http://localhost" or "http://localhost:XXXX" where "XXXX" is the actual port number.

The web based setup wizard has to be run after the installation is finished (it will automatically start the first time the site is browsed). You should be able to follow the instructions on the screen. Refer to "The setup wizard" (page 58) section for a complete step-by-step guide.

1.2.3 Installing an eZ publish bundle on Windows

This section describes how to deploy an eZ publish bundle on a Windows based operating system.

Important note about services

When installing under Windows, you have to be sure that there are no services running/listening on port 80 and 3306 (which are the default ports for HTTP and MySQL traffic). Services listening on these ports should be stopped. If Apache or MySQL is already installed, then the installer will replace these services with the services that are included in the bundle. Existing installations of Apache and MySQL will not be deleted or removed, only the services will be stopped and replaced.

Downloading the eZ publish bundle

The eZ publish bundle can be downloaded from http://ez.no/download/ez_publish. The bundle for Windows (95/98/ME/NT/2000/XP) is called something like "ezpublish-3.7.x_installer.exe".

Starting the installation wizard

When installing on Windows NT, 2000 or XP, make sure you're logged in with a user that has administrator privileges (preferably the "Administrator" user). Start the install wizard by clicking on its icon (double or single click depending on your settings).

Welcome dialog

(see figure 1.7)

This is the initial screen. Click "Next" to continue.

Component selection

(see figure 1.8)

The wizard will show a list of the components that will be installed. The installation of the PHP cache (MM Cache) is optional. Click "Next" to continue.

Destination folder

(see figure 1.9)

Choose a destination folder, everything will be put in here. Click "Next" to continue.



Figure 1.7: Step 1: Welcome dialog

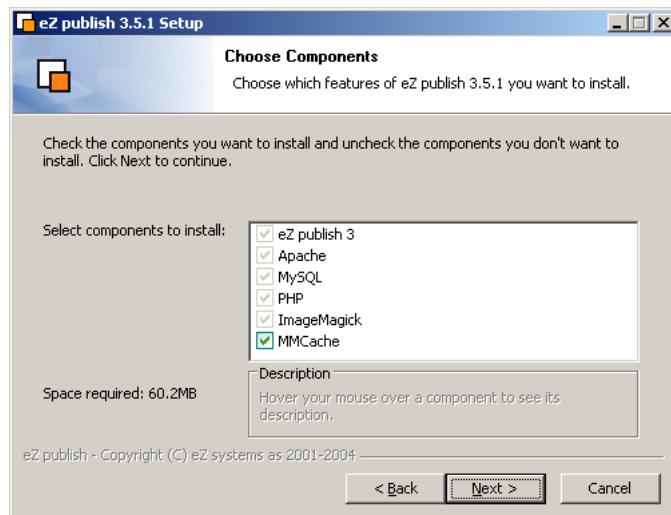


Figure 1.8: Step 2: Component selection

System information

Start menu

(see figure 1.10)

Choose a start menu folder for the program shortcuts. You may also choose not to create the shortcuts. Click "Next" to continue.

Actual installation

(see figure 1.11)

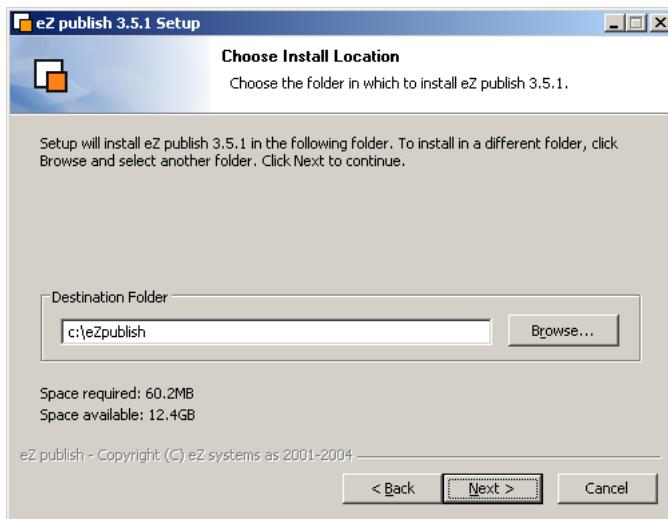


Figure 1.9: Step 3: Destination folder

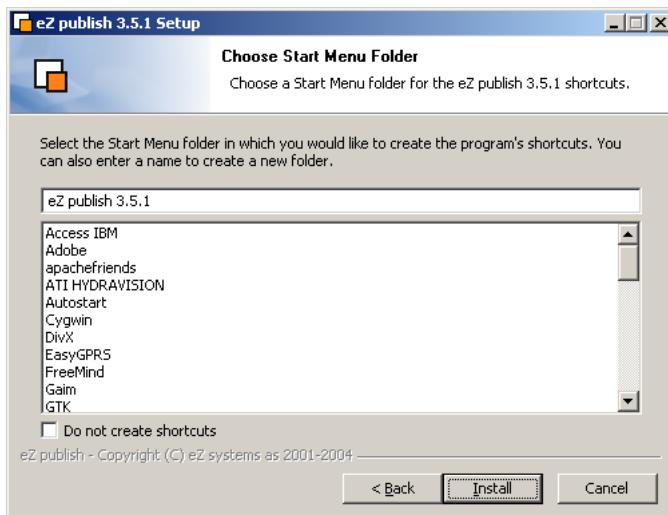


Figure 1.10: Step 4: Start menu

S
y
st
er
information

Please wait while the wizard copies/install the files.

The end

(see figure 1.12)

This is the final screen. The installation has finished. Click "Finish" to end/close the wizard. The eZ publish setup wizard has to be run after the installation is finished (it will automatically start the first time the site is browsed). The setup wizard is self-explaining and is used to configure the eZ publish system. You should be able to follow the instructions on the screen. Please refer

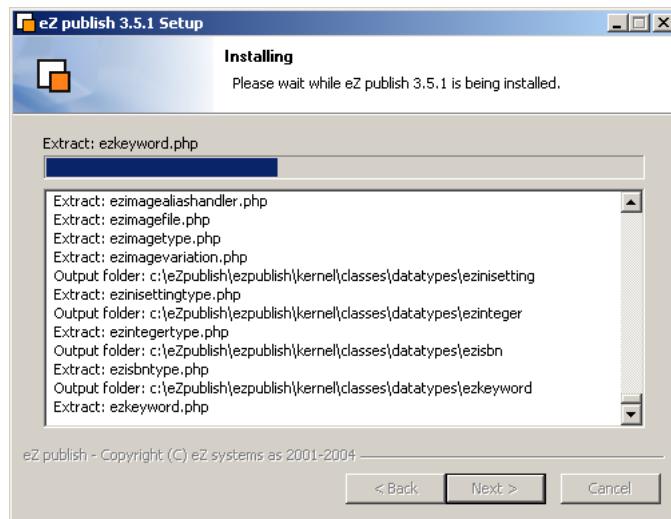


Figure 1.11: Step 5: Installation in progress



Figure 1.12: Step 6: Completion

to "The setup wizard" (page 58) section for more information.

1.3 Manual installation

This installation method is for advanced users who know what they are doing, all other users should use the "Normal installation method" (page 21). The manual installation method requires an environment which already has a web server, a database and etc. setup and ready to go; eZ publish needs to be downloaded and unpacked. Instead of running the setup wizard, all configuration is done manually using the command line interface of the target operating system. The following sections (depending on the target OS) will take you through the necessary steps.

your
information

1.3.1 Requirements for doing a manual installation

The requirements for doing a manual installation are the same as for the normal installation. Please refer to the "Requirements for doing a normal installation" (page [22](#)) section.

Your information

1.3.2 Manual installation on a Linux/UNIX based system

The requirements for doing a manual installation must be met. Please read the previous section if you're not sure about the requirements. Proceed only if you have access to a UNIX based environment with Apache, PHP, MySQL or PostgreSQL already installed and running. As mentioned earlier, the database server may run on a different computer than the web server. A manual installation consists of the following steps:

- Setting up a database (MySQL or PostgreSQL)
- Downloading eZ publish
- Unpacking eZ publish
- Manual configuration of eZ publish

The only difference between a normal and a manual installation is the last step. Instead of running the web based setup wizard, eZ publish is manually configured by editing a couple of files. The first three steps are explained under the "Installing eZ publish on a Linux/UNIX based system" (page 25) section. The last step is explained under the "Manual configuration of eZ publish" (page 46) section.

1.3.3 Manual installation on Windows

The requirements for doing a manual installation must be met. Please read the previous section if you're not sure about the requirements. Proceed only if you have access to a Windows based system with Apache, PHP, MySQL or PostgreSQL already installed and running. As mentioned earlier, the database server may run on a different computer than the web server. A manual installation consists of the following steps:

- Setting up a MySQL database
- Downloading eZ publish
- Unpacking eZ publish
- Manual configuration of eZ publish

The only difference between a normal and a manual installation is the last step. Instead of running the web based setup wizard, eZ publish is manually configured by editing a couple of files. The first three steps are explained under the "Installing eZ publish on Windows" (page 28) section. The last step is explained under the "Manual configuration of eZ publish" (page 46) section.

1.3.4 Manual configuration of eZ publish

This section describes how to manually configure eZ publish instead of using the setup wizard to do all the work. Please keep in mind that the manual installation method is for expert users only. It should only be used by people who know what they are doing. The following steps will work on both Linux/UNIX and Windows environments.

Database initialization

A clean eZ publish database is created using two very important SQL scripts: "kernel_schema" and "cleandata". While the "kernel_schema" script differs for each database engine, the "cleandata" script is the same for all solutions.

MySQL

Use the following command to run the MySQL specific "kernel_schema" script:

```
$ mysql -u <username> -p<password> <database> < /path/to/ez_publish/kernel/sql/mysql/kernel_schema.sql
```

Note that the CREATE TABLE statements in the "kernel_schema" script do not specify which storage engine to use (no ENGINE or TYPE option), and thus the default storage engine will be used. Normally, it is MyISAM. If you are running MySQL 4.x or later, it is recommended to use the InnoDB engine instead (if it is available on your server). To do this, set the default storage engine to InnoDB before you run the "kernel_schema" script (refer to the [MySQL documentation](#) for information about how to set the default engine).

When installing an eZ Publish version downloaded from a subversion repository, "TYPE=MyISAM" is explicitly specified in "kernel_schema.sql". To make the script use the InnoDB storage engine, replace all occurrences of "TYPE=MyISAM" with "TYPE=InnoDB" before you run the script. In addition, set the default storage engine to InnoDB, otherwise future upgrades might leave you with a mix of table types.

Alternatively, you can run the "kernel_schema" script first and then convert the newly created tables to InnoDB. You can either use the "bin/php/ezconvertmysqltabletype.php" script for database conversion (recommended) or convert the tables individually by using the following SQL query for each table:

```
ALTER TABLE <name_of_table> TYPE = innodb;
```

Use the following command to run the generic "cleandata" script:

```
$ mysql -u <username> -p<password> <database> < /path/to/ez_publish/kernel/sql/common/cleandata.sql
```

username

The MySQL user (if no user is set up, use

	”root”).
password	The password that belongs to the username.
database	The name of the database, for example ”my_database”.

File permissions

Windows users can skip this part. If eZ publish is installed on a Linux/UNIX based system, some of the file permissions need to be changed. There exists a shell script that takes care of this. This script must be run, or else, eZ publish will not function properly. The script needs to be run from within the eZ publish directory:

```
$ cd /path/to/ez_publish
$ bin/modfix.sh
```

The modfix script recursively alters the permission settings of the following directories:

- var/*
- settings/*
- design/*

If you know the user and group of the webserver it is recommended to use a different set of permissions:

```
$ chmod og+rwx -R var
$ chown -R example.example var
```

The ”example.example” notation must be changed to user and groupname of the webserver.

Configuring eZ publish

The ”/path/to/ez_publish/settings/override/site.ini.append.php” configuration file must be changed, or else eZ publish will not function properly. There are a lot of things that need to be configured (database, mail transport system, var directory, etc.). The following text shows a generic example of a configuration that can be used:

```
<?php /* #?ini charset="iso-8859-1"?>

[DatabaseSettings]
DatabaseImplementation=ezmysql
Server=localhost
User=root
```

```
1  
S  
e  
z  
p  
u  
b  
l  
i  
s  
h  
  
Password=  
Database=example  
  
[FileSettings]  
VarDir=var/example  
  
[Session]  
SessionNameHandler=custom  
  
[SiteSettings]  
DefaultAccess=example  
SiteList[]  
SiteList []=example  
  
[SiteAccessSettings]  
CheckValidity=false  
AvailableSiteAccessList[]  
AvailableSiteAccessList []=example  
AvailableSiteAccessList []=example_admin  
MatchOrder=host;uri  
  
# Host matching  
HostMatchMapItems []=www.example.com;example  
HostMatchMapItems []=admin.example.com;example_admin  
  
[InformationCollectionSettings]  
EmailReceiver=webmaster@example.com  
  
[MailSettings]  
Transport=sendmail  
AdminEmail=webmaster@example.com  
EmailSender=test@example.com  
  
[RegionalSettings]  
Locale=eng-GB  
ContentObjectLocale=eng-GB  
TextTranslation=disabled  
  
*/ ?>
```

In addition, two siteaccess configurations must be created, a public siteaccess and an administration siteaccess. In the example above "example" and "example_admin" is used. The following directories have to be created:

- /path/to/ez_publish/settings/siteaccess/example

- /path/to/ez_publish/settings/siteaccess/example_admin

Both siteaccesses must have a file called "site.ini.append.php".

The public siteaccess

The following text shows a generic solution for the "example" siteaccess:

```
<?php /* #?ini charset="iso-8859-1"?>

[SiteSettings]
SiteName=Example
SiteURL=www.example.com
LoginPage=embedded

[SiteAccessSettings]
RequireUserLogin=false
ShowHiddenNodes=false

[DesignSettings]
SiteDesign=example

[ContentSettings]
ViewCaching=disabled

[TemplateSettings]
TemplateCache=disabled
TemplateCompile=disabled
#ShowXHTMLCode=enabled
#Debug=enabled

[DebugSettings]
DebugOutput=enabled
Debug=inline
#DebugRedirection=enabled

*/ ?>
```

The admin siteaccess

The following text shows a generic solution for the "example_admin" siteaccess:

```
<?php /* #?ini charset="iso-8859-1"?>
```

```
[SiteSettings]
SiteName=Example
SiteURL=admin.example.com
LoginPage=custom

[SiteAccessSettings]
RequireUserLogin=true
ShowHiddenNodes=true

[DesignSettings]
SiteDesign=admin

[ContentSettings]
CachedViewPreferences[full]=admin_navigation_content=0;
admin_navigation_details=0;admin_navigation_languages=0;
admin_navigation_locations=
0;admin_navigation_relations=0;admin_navigation_roles=0;
admin_navigation_policies=0;admin_navigation_content=0;
admin_navigation_translatio
ns=0;admin_children_viewmode=list;admin_list_limit=1;
admin_edit_show_locations=0;admin_url_list_limit=10;admin_url_view_limit=10;
admin_sec
tion_list_limit=1;admin_orderlist_sortfield=user_name;
admin_orderlist_sortorder=desc;admin_search_stats_limit=1;admin_treemenu=1;
admin_boo
kmarkmenu=1;admin_left_menu_width=13

[DebugSettings]
DebugOutput=disabled
Debug=inline

*/ ?>
```

Unicode support

If you're using a database which supports Unicode (for example MySQL 4.1.x or later) and PHP is compiled with multibyte string support, create the following file: "/path/to/ez_publish/settings/override/i18n.ini.append.php" and make sure that it contains these lines:

```
<?php /* #?ini charset="iso-8859-1"?>

[CharacterSettings]
Charset=utf-8
MBStringExtension=enabled
```

```
*/ ?>
```

1

Primary language

The default primary language (used by the "cleandata" SQL script) is British English (eng-GB). If you're planning to set up a site where the primary language should be something else than British English, you'll have to change this by running a couple of SQL commands. The following examples shows how to switch the primary to Norwegian Bokmål (nor-NO).

Content object names

```
UPDATE ezcontentobject_name
SET
    content_translation='nor-NO',
    real_translation='nor-NO'
WHERE
    content_translation='eng-GB' OR
    real_translation='eng-GB';
```

Content object attributes

```
UPDATE ezcontentobject_attribute
SET
    language_code='nor-NO'
WHERE
    language_code='eng-GB';
```

1.4 Automated installation

The automated installation method (also known as "kickstart") is for experienced users. It provides an automated version of the "Normal installation method" and is designed for system administrators who wish to roll out pre-configured installations of eZ publish. This method requires minimum interaction with the web based setup wizard and thus it can be used to rapidly deploy eZ publish on a massive scale. This method has the same requirements as the "Normal installation" method. A typical automated installation process consists of the following steps:

- Setting up / creating a database
- Downloading a packaged eZ publish distribution
- Unpacking the eZ publish distribution
- Configuring the "kickstart.ini" file
- Initiating the web based setup wizard

Once the web based setup wizard has completed, eZ publish will be ready for use.

1.4.1 Requirements for doing an automated installation

The requirements for an automated installation are the same as for the normal installation method. Please refer to the "Requirements for doing a normal installation" (page 22) page for more information.

At the minimum, a web server, a PHP engine, and a database server must be installed. Additional server-side software is only necessary if the kickstart configuration file instructs the system to make use of such software. For example, "ImageMagick" has to be available if it has been specified as the primary image manipulation solution.

The following section explains how eZ publish can be configured to do an automated installation of itself.

ez
information

1.4.2 Automated installation of eZ publish

The requirements for doing an automated installation must be met. Please read the previous section if you're not sure about the requirements. This section will guide you through the following steps:

- Setting up a database (MySQL or PostgreSQL)
- Downloading eZ publish
- Unpacking eZ publish
- Configuring the kickstart system
- Starting the installation by initiating the web based setup wizard

Depending on the target system, please refer to either "Installing eZ publish on a Linux/UNIX based system" (page 25) or "Installing eZ publish on Windows" (page 28) for information about the first three steps (database setup, download and unpacking). The rest of the steps are explained below.

Configuring the kickstart system

The behavior of the automated installation is controlled by the "kickstart.ini" configuration file. This file makes it possible to specify parameters for each installation step of the web based setup wizard. For example, by providing the database connection parameters, the corresponding setup wizard step will have the input forms pre-filled. It is also possible to instruct the wizard to skip certain steps.

Initialization

Create a copy of the "kickstart.ini-dist" file (located in the root of your eZ publish installation) and make sure that the copy is named "kickstart.ini" (located in the root of eZ publish). The following example shows how this can be done on a Linux/UNIX based system:

1. Navigate into the eZ publish directory:

```
$ cd /path/to/ezpublish/
```

2. Copy and rename the configuration file:

```
$ cp kickstart.ini-dist kickstart.ini
```

Security issues

The web server must have read access to the "kickstart.ini" file during the installation process. This might become a security problem at a later stage if the file contains usernames, passwords, etc. To prevent this from happening, it is recommended to do one of the following:

- Remove the file when the installation has completed.
- Use rewrite rules to make sure that it is not readable from outside.

Configuration blocks

The "kickstart.ini" file contains a configuration block for every step of the setup wizard. The block names are encapsulated by square brackets. The following list shows an overview of the available blocks.

- [email_settings]
- [database_choice]
- [database_init]
- [language_options]
- [site_types]
- [site_packages]
- [site_access]
- [site_details]
- [site_admin]
- [security]
- [registration]

In the default kickstart file, everything is commented out. The blocks and the corresponding settings have to be uncommented in order to take effect. This can be done by removing the hash ("#") characters from the start of the lines that you should be activated. Make sure that there are no leading whitespace characters at the start of the lines.

Configuration parameters

Each parameter takes a text string as an input value. Some parameters are able to handle an array of strings. The following examples demonstrate the two parameter types.

- Single parameter:

```
Server=www.example.com
```

- Array parameter:

```
Title[]
Title[news]=The news site
Title[forums]=The forum site
```

Documentation and examples

The "kickstart.ini" file contains documentation in the file itself. Please refer to the embedded instructions and examples for a detailed explanation of the steps. The following table shows how the examples / inline instructions deal with required and optional parameters.

Syntax	Description
<value>	Angle brackets indicate that the parameter value is required, example: <code>#Server=<hostname></code>
[value]	Squared brackets indicate that the parameter value is optional, example: <code>#FirstName=[string]</code>

A parameter will only take effect if it has been uncommented. Remove the leading hash ("#") and make sure that there are no whitespace characters at the start of the lines that include the uncommented parameters.

Skipping steps

A step can be skipped by uncommenting and setting the "Continue" parameter to "true". This parameter can be used for each step / block. The following table shows the outcome for the different configurations of the "Continue" parameter.

Assignment	Result
Continue=false	The step will be shown and the input values will be pre-filled with the values (if any) defined in the "kickstart.ini" configuration file. This is the same as when the "Continue" parameter is missing or if it has been commented out.
Continue=true	The system will automatically use the values defined in the kickstart file and thus the step

will not be shown. However, if something goes wrong (missing or wrong values, etc.), the step will be shown.

Starting the installation

The installation can be started by initiating the web based setup wizard. Please refer to the "Initiating the setup wizard" part of the "Normal installation" section.

1.5 The setup wizard

This section contains a comprehensive guide through the web based setup wizard of eZ Publish. The setup wizard is designed to ease the initial configuration of the system. It can be started using a web browser when the necessary installation steps (described in the previous sections) are completed. The setup wizard will automatically start the first time the "index.php" file (located in the root of the eZ Publish directory) is accessed/browsed.

The setup wizard does not store or modify any data before the final step; thus, it can be safely restarted by reloading the URL containing only the "index.php" part. The back button (located at the bottom) can be used to jump back to previous steps in order to modify settings. A typical setup cycle consists of 13 steps:

1. Welcome page
2. System check
3. Outgoing E-mail
4. Database choice (optional)
5. Database initialization
6. Language support
7. Site type
8. Site functionality
9. Access method
10. Site details
11. Site security
12. Site registration
13. Finish

Please note that some of the steps will be omitted when an eZ Publish bundle (page 31) is being installed.

Welcome page

(see figure 1.13)

This is the initial page of the setup wizard. By clicking "Next", the wizard will either jump to the "System Check" page (if some issues need to be fixed) or to the "Outgoing E-mail" page (if everything is okay).

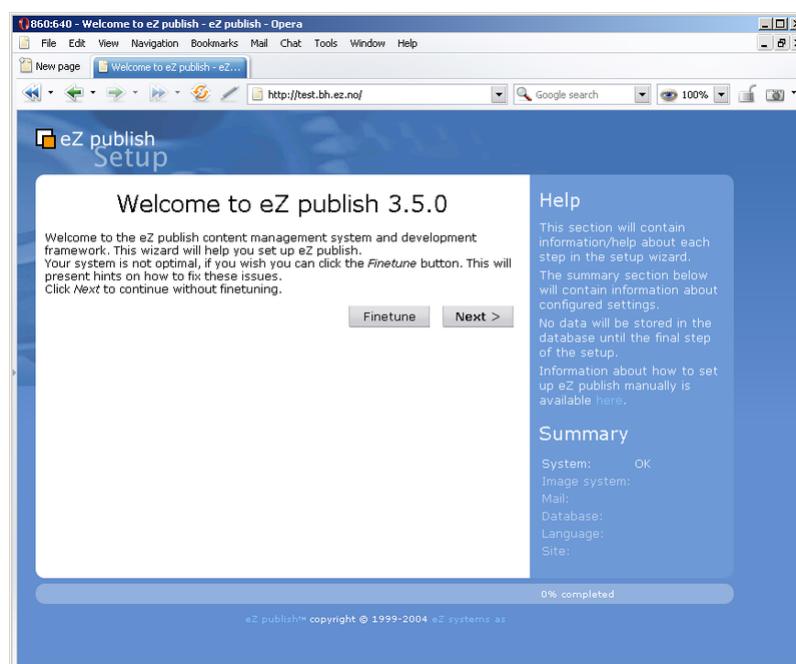


Figure 1.13: Step 1: Welcome page

System check

(see figure 1.14)

This page usually appears if critical issues/problems are detected. The setup wizard will display information about the issues that need to be fixed and suggestions describing how they can be fixed.

Issues

There may be several issues/problems. A suggestion to each problem is presented below the description of the problem itself. The setup wizard will probably suggest the execution of miscellaneous shell commands (in order to fix ownerships, permissions, etc.). These commands must be executed using a system shell. Simply copy the commands from the browser window and paste them into an open shell. The setup wizard will run the system check again when the "Next" button is clicked. The "System check" page will keep reappearing until all issues have been fixed (or ignored, see the next section). Once everything is okay, the setup wizard will display the next step.

Ignoring tests

Some issues/problems may be ignored using a checkbox labelled "Ignore this test". However, it is recommended to fix all issues rather than ignoring them.

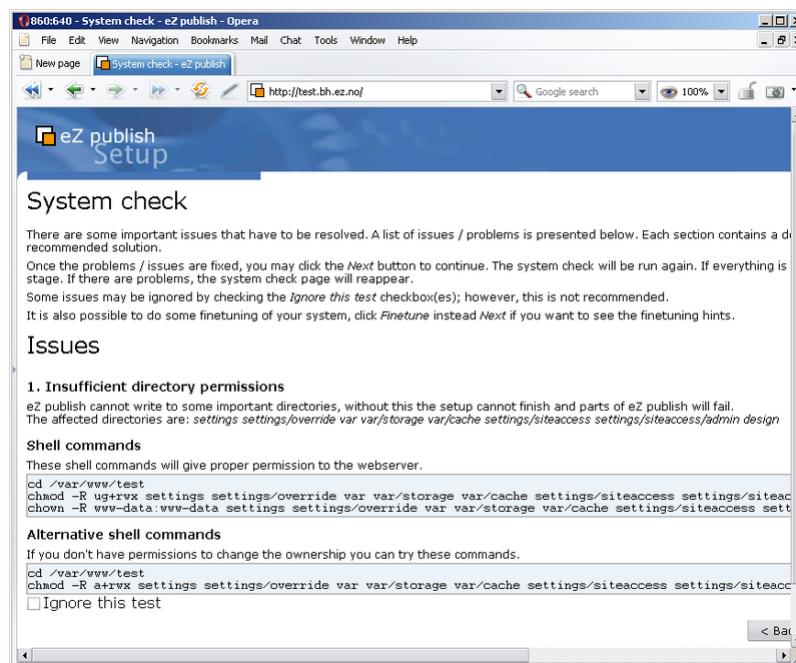


Figure 1.14: Step 2: Issues

Outgoing E-mail

(see figure 1.15)

eZ Publish uses E-mail to send out miscellaneous notices. This step is used to configure how eZ Publish delivers outgoing E-mail. There are two options:

- Direct delivery through sendmail (must be available on the server)
- Indirect delivery using an SMTP (Simple Mail Transfer Protocol) relay server

On Linux/UNIX: try to use sendmail; use SMTP if sendmail is unavailable. On Windows: use the SMTP setting.

Sendmail

Mail is delivered directly using the sendmail transfer agent. The agent must be running on the same host as the webserver is running on. The sendmail binary is usually available on most Linux/UNIX systems. If sendmail is not available then SMTP should be used.

SMTP

Mail is delivered through an SMTP server. At the minimum, the hostname of the SMTP server must be specified.

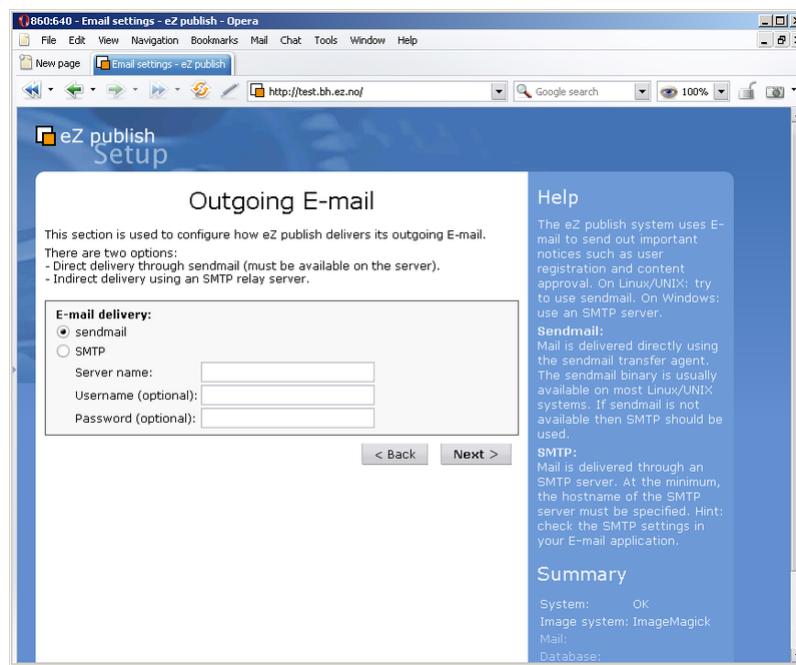


Figure 1.15: Step 3: Outgoing E-mail

Database type

(see figure 1.16)

The setup will automatically detect database support that has been made available for the PHP scripting engine. If both MySQL and PostgreSQL is supported, the database choice dialog will appear. If PHP only is setup only to support one type of database, eZ Publish will automatically use it and thus the database choice dialog will not be displayed.

Database initialization

(see figure 1.17)

Information about the hostname of the server running the database engine, and a username/password combination needs to be provided. When clicking "Next" and if MySQL is used, the setup wizard will attempt to connect to the database. The setup will only continue if it is able to connect to the specified MySQL server with the specified username/password combination. PostgreSQL parameters are tested at a later stage during the setup wizard.

Language support

(see figure 1.18)

This step allows the user to choose a language configuration for the site that is being installed. Only one primary language can be used. However, it is possible to use several additional lan-

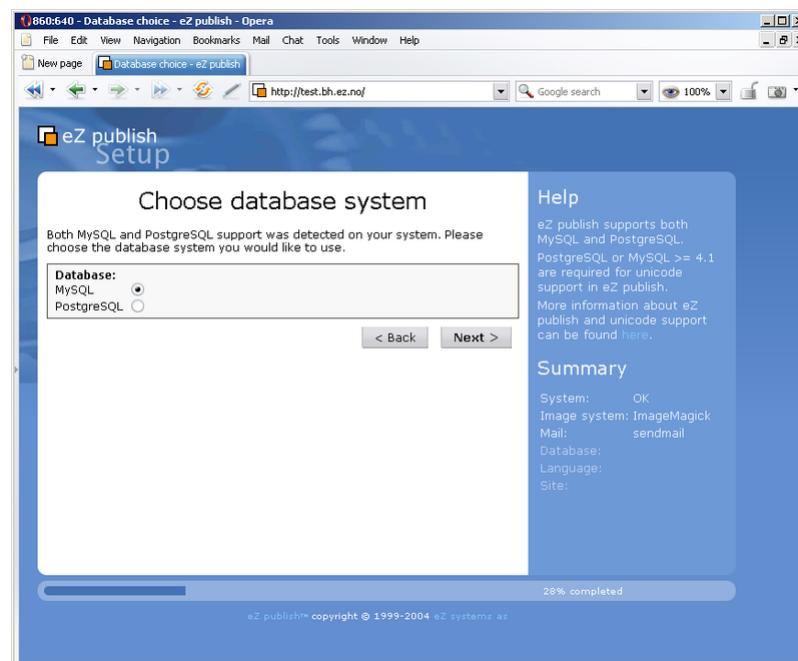


Figure 1.16: Step 4: Database choice

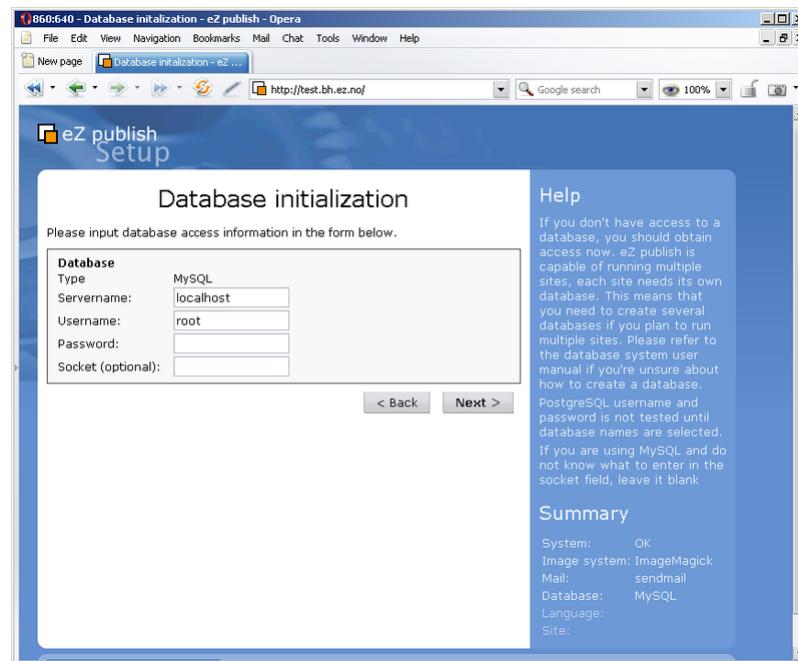


Figure 1.17: Step 5: Database initialization

guages. Additional languages allow the translation of content, miscellaneous date/number formats, the display of the administration interface using different languages and so on. The addi-

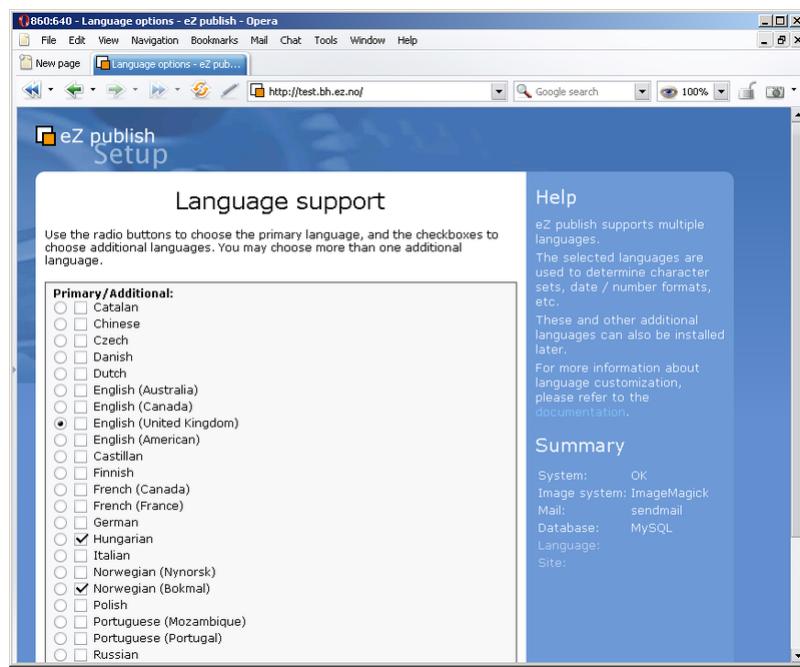


Figure 1.18: Step 6: Language support

tional languages can be reconfigured at any time (even when a site is up and running) using the administration interface.

Site type

(see figure 1.19)

This step allows the user to select one of the built-in site types. eZ Publish comes with a collection of basic examples (News, Company, Intranet, Gallery, etc.). These examples are mostly for the purpose of demonstration and learning. However, it is possible to use them as a basic framework which you can extend/tweak in order to make it suitable for a specific purpose. A demo site usually contains some artwork (images), CSS code, actual content and template files. The plain type should be used when starting from scratch.

Site functionality

(see figure 1.20)

This step allows the selection of additional features / demo data that should be installed. These packages can also be installed at a later stage using the administration interface.

Access method

(see figure 1.21)

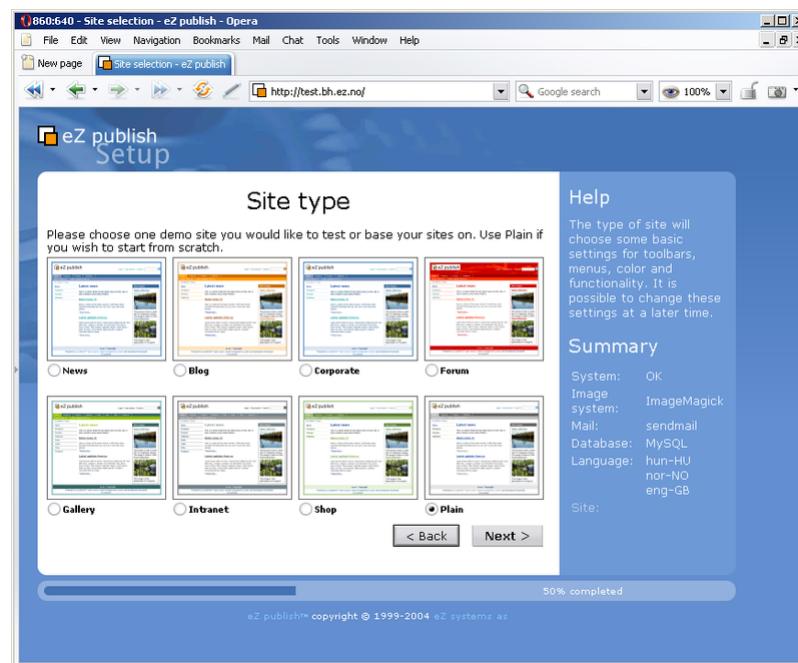


Figure 1.19: Step 7: Site type

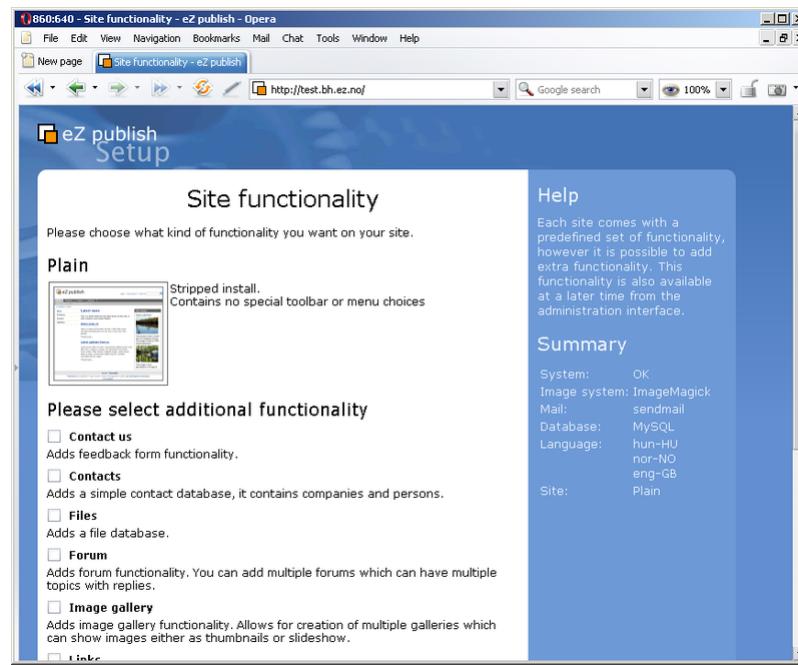


Figure 1.20: Step 8: Site functionality

This step allows the configuration of the access method that should be used when eZ Publish receives a request. There are three options:

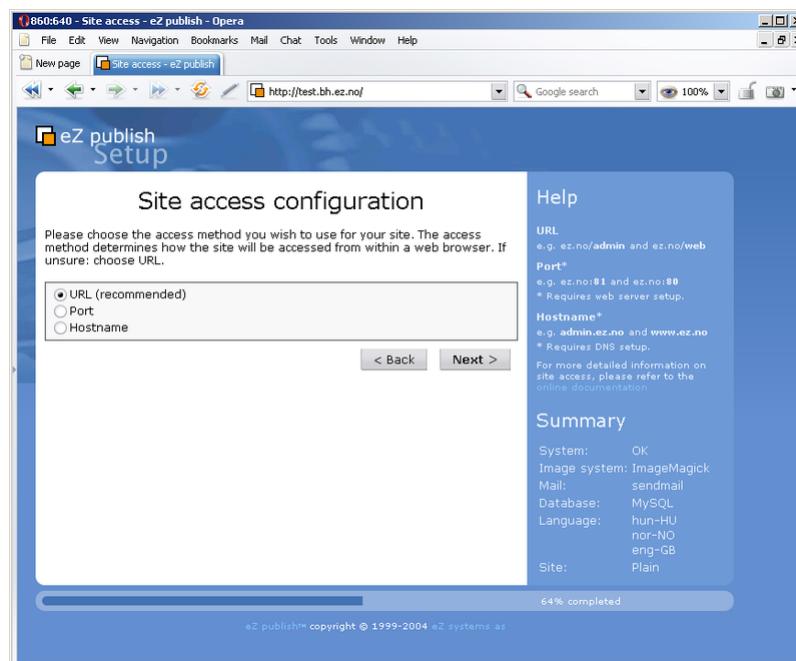


Figure 1.21: Step 9: Site access configuration

- URL
- Port
- Hostname

URL

When the URL access method is used, eZ Publish selects the site that should be accessed based on the contents of the URL (in particular the part that comes right after "index.php"). This is the default and most generic option. It doesn't require any additional configuration. Use this setting when installing eZ Publish for the first time.

Port

When the port access method is used, eZ Publish selects the site that should be accessed based on a port number that is specified in the URL. The port number must be appended to the hostname of the web server: "http://www.example.com:81/index.php". This option requires additional web server and firewall configuration. Use this setting only if you know what you're doing.

Hostname

When this access method is used, each site is assigned a unique hostname. For example, "www.example.com" and "admin.example.com" can be assigned to the public and the administration interface respectively. This option requires additional web and DNS server configuration. Use this setting only if you know what you're doing.

Site details

(see figure 1.22)

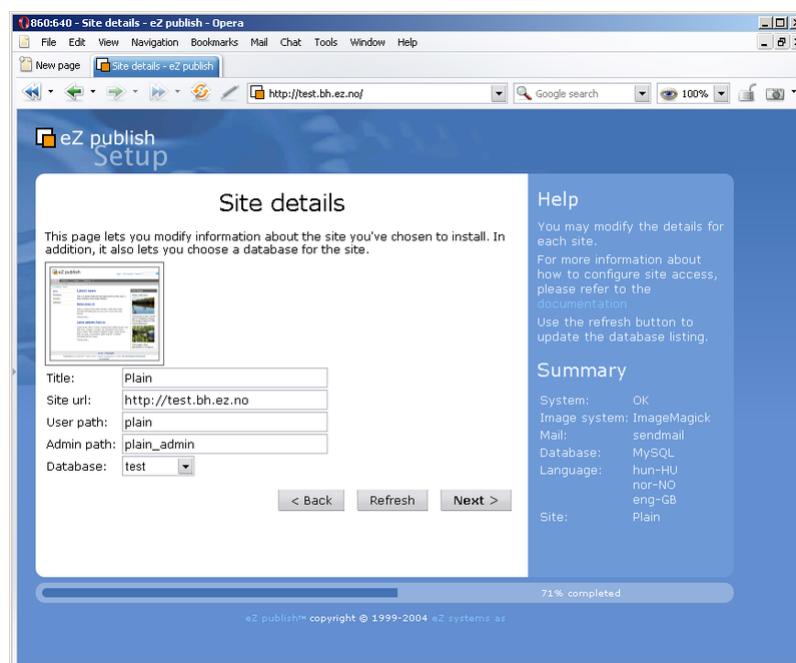


Figure 1.22: Step 10: Site details

This step allows the modification of settings related to the site that is being installed. Please note that the "User path" and "Admin path" access values depend on which access method you choose. When the port access method is used these values are port numbers. If you use the URL access method then "User path" and "Admin path" should only contain letters, digits and underscores. If the hostname access method is used then some additional symbols like dashes, dots and colons are allowed whereas underscores aren't.

The available databases will be displayed in the database dropdown menu. The "Refresh" button can be used to update the list (if a database is being created at this point).

If the selected database already contains data, the "Site Details" page will reappear and ask what to do. Possible actions are:

- Leave the data and add new

- Remove existing data
- Leave the data and do nothing
- I've chosen a new database

Use the last option if another database has been chosen.

Site security

(see figure 1.23)

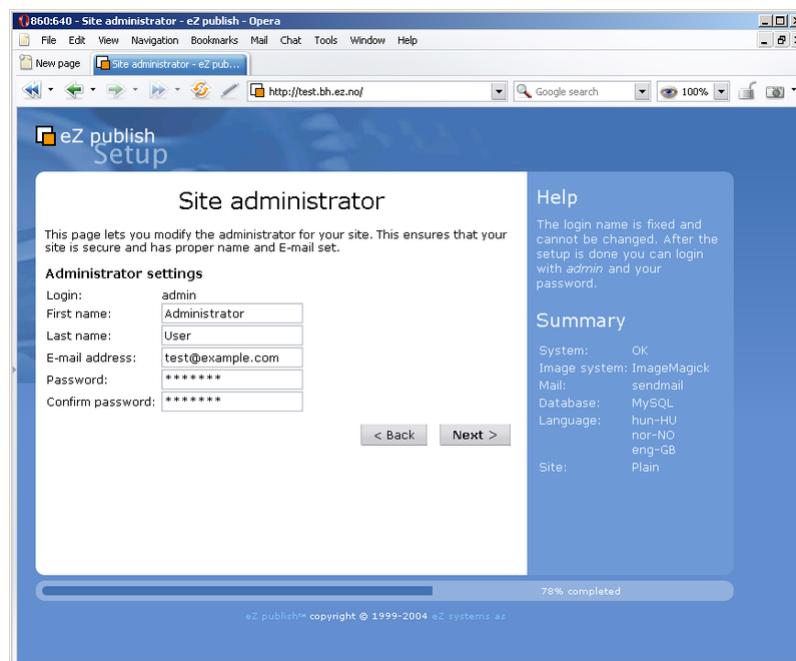


Figure 1.23: Step 11: Site administrator

This step suggests some basic modifications that should be carried out in order to secure the site being installed. The suggested security tweak protects the configuration files from unwanted access. Don't worry about this unless you're setting up a site for public use.

Please note that the administrator's username (login) is set to "admin" by default and can not be changed. If you need another username for site administrator, you can install eZ Publish, create a new administrator user, log in as this user and remove the old one.

Site registration

(see figure 1.24)

This step allows you to control whether the setup should send an information E-mail to eZ Systems or not. The information will be used internally for statistics and for improving eZ Publish.

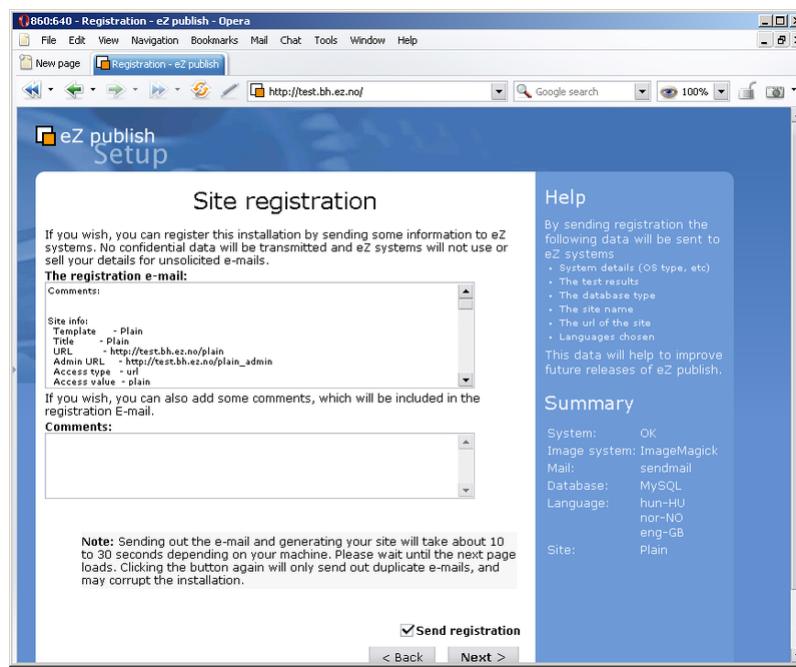


Figure 1.24: Step 12: Site registration

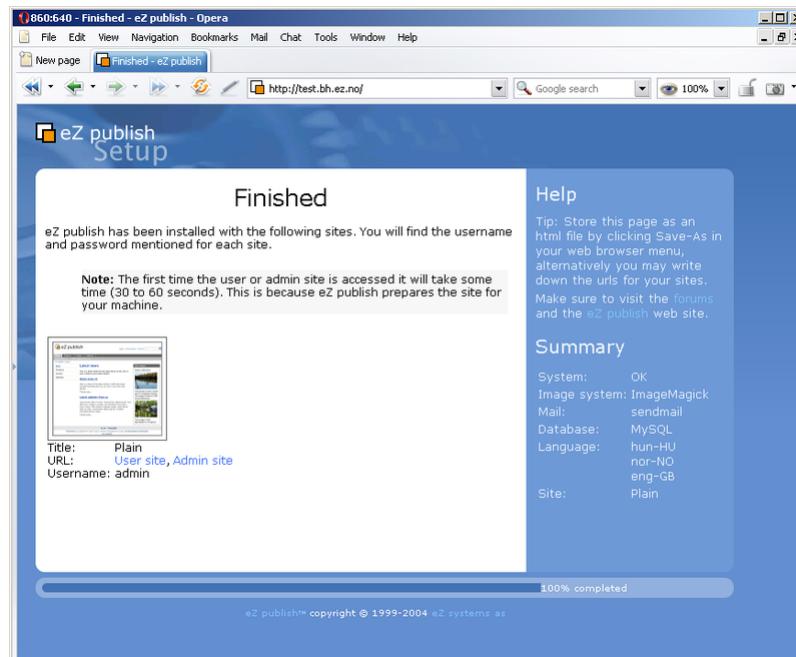
No confidential data will be transmitted and eZ Systems will not misuse or sell these details. The following information will be sent:

- System details (OS type, etc)
- The test results
- The type of database that is being used
- The name of the site
- The URL of the site
- The languages that were chosen

Finished

(see figure 1.25)

The setup wizard has finished, eZ Publish is ready for use. Click on one of the links to access the various interfaces (public site, administration interface, etc.).

Figure 1.25: Step 13: *Finished*

1.6 Virtual host setup

This section describes how to set up a virtual host for eZ Publish using the Apache webserver. A virtual host setup is only needed if eZ Publish has been configured to use the host access method, which is the most secure method.

By making use of virtual hosts, it is possible to have several sites running on the same server. The sites are usually differentiated by the name they are accessed. Apache will look for a specified set of domains and use different configuration settings based on the domain that is accessed.

Generic virtual host setup

Virtual hosts are usually defined at the end of "httpd.conf", which is the main configuration file for Apache. Adding a virtual host for eZ Publish can be done by copying the following lines and replacing the text encapsulated by the square brackets with actual values. Please refer to the next section for a real life example of using virtual hosts.

```
NameVirtualHost [IP_ADDRESS]

<VirtualHost [IP_ADDRESS]:[PORT]>
    <Directory [PATH_TO_EZPUBLISH]>
        Options FollowSymLinks
        AllowOverride None
    </Directory>

    <IfModule mod_php4.c>
        php_admin_flag safe_mode Off
        php_admin_value register_globals 0
        php_value magic_quotes_gpc 0
        php_value magic_quotes_runtime 0
        php_value allow_call_time_pass_reference 0
    </IfModule>

    DirectoryIndex index.php

    <IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteRule ^/var/storage/.* - [L]
        RewriteRule ^/var/[^/]+/storage/.* - [L]
        RewriteRule ^/var/cache/texttoimage/.* - [L]
        RewriteRule ^/var/[^/]+/cache/texttoimage/.* - [L]
        RewriteRule ^/design/[^/]+/(stylesheets|images|javascript)/.* - [L]
        RewriteRule ^/share/icons/.* - [L]
        RewriteRule ^/extension/[^/]+/design/[^/]+/(stylesheets|images|javascripts?)/.* - [L]
```

```

Rewriterule ^/packages/styles/.+/(stylesheets|images|javascript)/[^/]+/
.* - [L]
    RewriteRule ^/packages/styles/.+/thumbnail.* - [L]
    RewriteRule ^/favicon.ico - [L]
    RewriteRule ^/robots.txt - [L]
    # Uncomment the following lines when using popup style debug.
    # RewriteRule ^/var/cache/debug\.html.* - [L]
    # RewriteRule ^/var/[^/]+/cache/debug\.html.* - [L]
    RewriteRule .* /index.php
</IfModule>

DocumentRoot [PATH_TO_EZPUBLISH]
ServerName [SERVER_NAME]
ServerAlias [SERVER_ALIAS]

</VirtualHost>

```

[IP_ADDRESS]	The IP address of the virtual host, for example "128.39.140.28". Apache allows the usage of a wildcards here ("*").
[PORT]	The port on which the webserver listens for incoming requests. This is an optional setting, the default port is 80. The combination of an IP address and a port is often referred to as a socket. Apache allows the usage of a wildcards here ("*").
[PATH_TO_EZPUBLISH]	Path to the directory that contains eZ Publish. This must be the full path, for example "/var/www/ezpublish-3.6.0".
[SERVER_NAME]	The host or the IP address that Apache should look for. If a match is found, the virtual host settings will be used.
[SERVER_ALIAS]	Additional hosts/IP addresses that Apache should look for. If a match is found, the virtual host settings will be used.

Please note that the "mod_rewrite" module must be enabled in "httpd.conf" in order to use the Rewrite Rules.

NameVirtualHost

The "NameVirtualHost" setting might already exist in the default configuration. Defining a new one will result in a conflict. If Apache reports errors such as "NameVirtualHost [IP_ADDRESS] has no VirtualHosts" or "Mixing * ports and non-* ports with a NameVirtualHost address is not supported", try skipping the NameVirtualHost line. For more info about the NameVirtualHost directive, see <http://httpd.apache.org/docs/1.3/mod/core.html#namevirtualhost>.

SOAP and WebDAV

If you would like to use the SOAP and/or the WebDAV features of eZ Publish, you'll have to add the following lines in the virtual host configuration:

```
RewriteCond %{HTTP_HOST} ^webdav\..*
RewriteRule ^(.*) /webdav.php [L]

RewriteCond %{HTTP_HOST} ^soap\..*
RewriteRule ^(.*) /soap.php [L]

ServerAlias soap.example.com
ServerAlias webdav.example.com
```

1.6.1 Virtual host example

This example demonstrates how to set up a virtual host on the Apache web server for an eZ publish installation located in ”/var/www/example”. Let’s say that we want to access eZ publish by using the following URLs:

- <http://www.example.com> (actual website for public access)
- <http://admin.example.com> (administration interface for the webmaster)

In order to achieve this, we need to set up both eZ publish and the web server so that they respond correctly to the different requests.

eZ publish configuration: siteaccess settings

eZ publish needs to be configured to use the host access method. This can be done from within the web based setup wizard or by manually editing the global override for the site.ini configuration file: ”/settings/override/site.ini.append.php”. A typical configuration would look something like this:

```
...
[SiteAccessSettings]
AvailableSiteAccessList []
AvailableSiteAccessList []=example
AvailableSiteAccessList []=example_admin
MatchOrder=host

HostMatchMapItems []=www.example.com;example
HostMatchMapItems []=admin.example.com;example_admin
...
```

This configuration tells eZ publish that it should use the ”example” siteaccess if a request starts with ”www.example.com” and ”example_admin” if the request starts with ”admin.example.com”. For more information about site management in eZ publish, please refer to the ”Site management” (page 142) part of the ”Concepts and basics” chapter.

Apache configuration: virtual host settings

Assuming that...

- eZ publish is located in ”/var/www/example”
- the server’s IP address is 128.39.140.28
- we wish to access eZ publish using ”www.example.com” and ”admin.example.com”

...the following virtual host configuration needs to be added at the end of "http.conf":

```
NameVirtualHost 128.39.140.28

<VirtualHost 128.39.140.28>
    <Directory /var/www/example>
        Options FollowSymLinks
        AllowOverride None
    </Directory>

    <IfModule mod_php4.c>
        php_admin_flag safe_mode Off
        php_admin_value register_globals      0
        php_value magic_quotes_gpc          0
        php_value magic_quotes_runtime     0
        php_value allow_call_time_pass_reference 0
    </IfModule>

    DirectoryIndex index.php

    <IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteRule ^/var/storage/.* - [L]
        RewriteRule ^/var/[^\]+/storage/.* - [L]
        RewriteRule ^/var/cache/texttoimage/.* - [L]
        RewriteRule ^/var/[^\]+/cache/texttoimage/.* - [L]
        RewriteRule ^/design/[^\]+/(stylesheets|images|javascript)/.* - [L]
        RewriteRule ^/share/icons/.* - [L]
        RewriteRule ^/extension/[^\]+/design/[^\]+/
(stylesheets|images|javascripts?)/.* - [L]
        RewriteRule ^/packages/styles/.+/(stylesheets|images|javascript)/[^\]+/
.* - [L]
        RewriteRule ^/packages/styles/.+/thumbnail/.* - [L]
        RewriteRule ^/favicon.ico - [L]
        RewriteRule ^/robots.txt - [L]
        # Uncomment the following lines when using popup style debug.
        # RewriteRule ^/var/cache/debug\.html.* - [L]
        # RewriteRule ^/var/[^\]+/cache/debug\.html.* - [L]
        RewriteRule .* /index.php
    </IfModule>

    DocumentRoot /var/www/example
    ServerName www.example.com
    ServerAlias admin.example.com
</VirtualHost>
```

Please note that it isn't necessary to create a separate virtual host block for "admin.example.com", it can be added to the existing block using the "ServerAlias" directive.

You can have apache1 and apache2 part in the sample vhost. That way allows to use one vhost for both servers.

```
<IfModule mod_php4.c>
# If you are using Apache 2, you have to use <IfModule sapi_apache2.c>
# instead of <IfModule mod_php4.c>.
    # some parts/addons might only run safe mode on
    php_admin_flag safe_mode Off
    # security just in case
    php_admin_value register_globals 0
    # performance
    php_value magic_quotes_gpc 0
    # performance
    php_value magic_quotes_runtime 0
    #http://www.php.net/manual/en/
ini.core.php#ini.allow-call-time-pass-reference
    php_value allow_call_time_pass_reference 0
</IfModule>

<IfModule sapi_apache2.c>
# If you are using Apache 2, you have to use <IfModule sapi_apache2.c>
# instead of <IfModule mod_php4.c>.
    # some parts/addons might only run safe mode on
    php_admin_flag safe_mode Off
    # security just in case
    php_admin_value register_globals 0
    # performance
    php_value magic_quotes_gpc 0
    # performance
    php_value magic_quotes_runtime 0
    #http://www.php.net/manual/en/
ini.core.php#ini.allow-call-time-pass-reference
    php_value allow_call_time_pass_reference 0
</IfModule>
```

1.7 Upgrading

All the information related to the upgrade of an eZ publish solution has been moved to the "Upgrading" chapter.

1.7.1 from 3.5.2 to 3.7.0

This section describes how to upgrade your existing eZ Publish 3.5.2 installation to version 3.7.0. If you are upgrading from a version prior to eZ Publish 3.5.2, you need to first upgrade to 3.5.2 before you can upgrade to 3.7.0.

Make sure that you have a working backup of the site before you do the actual upgrade. The upgrade procedure consists of the following steps:

1. Upgrading the distribution files to 3.7.0
2. Upgrading the database to 3.7.0
3. Running the 3.6.0 upgrade scripts
4. Updating the system configuration
5. Clearing the caches

Step 1: Upgrading the distribution files

The easiest way to upgrade the distribution files is to unpack eZ Publish 3.7.0 to a directory and then copy the directories that contain site-specific files from the existing installation. Make sure that you copy the following directories:

- design/example
- design/example_admin
- var
- settings/siteaccess
- settings/override

Replace "example" and "example admin" with actual names used by your siteaccesses.

Custom extensions

If you are using custom extensions then the subdirectories inside the "extension" directory will also have to be copied. However, make sure that you do not overwrite any extensions that come with eZ Publish (for example the "PayPal" extension).

Step 2: Upgrading the database

The following text describes how a 3.5.2 database can be upgraded to 3.7.0.

MySQL

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the first database upgrade script:

```
mysql -u <username> -p<password> <database> < update/database/mysql/3.6/
dbupdate-3.5.2-to-3.6.0.sql
```

3. Run the second database upgrade script:

```
mysql -u <username> -p<password> <database> < update/database/mysql/3.7/
dbupdate-3.6.0-to-3.7.0.sql
```

Note that the CREATE TABLE statements in the database upgrade scripts do not specify which storage engine to use (no ENGINE or TYPE option), and thus the default storage engine will be used. Normally, it is MyISAM (starting from MySQL v.3.23). If you are using InnoDB, make sure the default storage engine is set to InnoDB before you run the database upgrade script (refer to [MySQL documentation](#) for information about how to set the default engine). If you were not able to change the MySQL configuration on your server, and the upgrade left you with a mix of table types, you can use the "bin/php/ezconvertmysqltabletype.php" script for database conversion. It is also possible to convert the newly created tables to InnoDB using ALTER TABLE statements as shown in the following example:

```
ALTER TABLE table_name1 TYPE = innodb;
ALTER TABLE table_name2 TYPE = innodb;
...
```

Replace "table_name1", "table_name2" with the actual names of the tables that need to be converted.

PostgreSQL

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the first database upgrade script:

```
psql -d <database> -U <dbowner> < update/database/postgresql/3.6/
dbupdate-3.5.2-to-3.6.0.sql
```

3. Run the second database upgrade script:

```
psql -d <database> -U <dbowner> < update/database/postgresql/3.7/
dbupdate-3.6.0-to-3.7.0.sql
```

Step 3: Running the 3.6.0 upgrade scripts

The 3.6.0 version of eZ Publish introduced a couple of new features that are used in 3.7 version also. In order to make sure that your site is compatible with these features, you'll have to run a couple of upgrade scripts.

Internal linking

In order to be compatible with the [new internal linking feature](#), you will have to run the "convertxmllinks.php" script. This script should be run for all siteaccesses that use different databases. If you only have a public and an administration siteaccess (which is the most typical/usual case), then you will only need to run the script for one of the siteaccesses.

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the script (replace <siteaccess> with the name of your siteaccess(es)):

```
php update/common/scripts/convertxmllinks.php -s <siteaccess>
```

Advanced related objects

In order to be compatible with the [new advanced related objects feature](#), you will have to run the "updaterelatedobjectslinks.php" script. This script should be run for all siteaccesses that use different databases. If you only have a public and an administration siteaccess (which is the most typical/usual case), then you will only need to run the script for one of the siteaccesses.

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the script (replace <siteaccess> with the name of your siteaccess(es)):

```
php update/common/scripts/updaterelatedobjectslinks.php -s <siteaccess>
```

Step 4: Updating the system configuration

Administration interface toolbar changes

The right section of the administration interface has been slightly changed in 3.6.0. It now uses the toolbar system and features a couple of new toolbars for developers (clear cache, debug control, etc.). In order to make it work, you will have to add the following settings in "toolbar.ini.append.php" for the siteaccesses that use the admin design:

```
[Toolbar]
AvailableToolBarArray []=admin_right
AvailableToolBarArray []=admin_developer

[Tool]
AvailableToolArray []=admin_current_user
AvailableToolArray []=admin_bookmarks
AvailableToolArray []=admin_clear_cache
AvailableToolArray []=admin_quick_settings

[Toolbar_admin_right]
Tool []
Tool []=admin_current_user
Tool []=admin_bookmarks

[Toolbar_admin_developer]
Tool []
Tool []=admin_clear_cache
Tool []=admin_quick_settings
```

Visibility of hidden nodes

In eZ Publish 3.6.0, the default value of the "ShowHiddenNodes" configuration directive has become "false". This means that hidden nodes will not be available on any siteaccess. It is recommended to add the following line to the "[SiteAccessSettings]" section of the "settings/siteaccess/example_admin/site.ini.append.php" configuration file (replace "example_admin" by the actual name of your admin siteaccess):

```
ShowHiddenNodes=true
```

This will instruct the system to show hidden nodes in the administration interface.

Enabling database transaction support (optional)

eZ Publish 3.6.0 introduced support for database transactions. This feature makes eZ Publish less vulnerable for database errors and inconsistencies due to aborted requests. It is highly recommended to make use of this feature. Transaction support is available for MySQL from version 4.1 and above and for all PostgreSQL and Oracle versions. MySQL users must first convert the database (see below).

MySQL table conversion

MySQL users have to perform an extra step when enabling transaction support. Make sure that you have MySQL 4.0 or later before attempting to perform this step. Previous versions of eZ

Publish use "MyISAM" type tables. Such tables do not support transactions. The database must be converted so that all tables use the "InnoDB" type instead of "MyISAM". This can be done using the "ezconvertmysqltabletype.php" script:

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the database conversion script; example:

```
bin/php/ezconvertmysqltabletype.php --host=localhost --user=arnold  
--password=secret --database=ezpublish --newtype=innodb
```

(You will have to provide your own values for the "host", "user", "password" and "database" parameters.)

It is also recommended to set the default storage engine to InnoDB (refer to [MySQL documentation](#) for information about how this can be done).

Enabling transaction support in eZ Publish

Each siteaccess which is configured to communicate with a database that uses "InnoDB" tables can be configured to use transactions. This can be done by adding "Transactions=enabled" within the "[DatabaseSettings]" block in "site.ini.append.php" for the target siteaccess(es). You can also add this line to the "settings/override/site.ini.append.php" file instead of doing it for each of the target siteaccesses (this way is recommended if all your siteaccesses use the same database).

Step 5: Clearing the caches

Whenever an eZ Publish solution is upgraded, all caches must be cleared in a proper way. This should be done from within a system shell:

1. Navigate into the eZ Publish 3.7.0 directory.
2. Run the clear cache script:

```
bin/shell/clearcache.sh --clear-all
```

Make sure that all caches are cleared. Sometimes the script is unable to clear caches because of restrictive file/directory permission settings. Make sure that all caches have been cleared by inspecting the contents of the various cache subdirectories within the "var" directory.

1.7.2 from 3.7.x to 3.7.y

This section describes how to upgrade your existing eZ Publish 3.7.x installation to version 3.7.y, for example from 3.7.0 to 3.7.9. If you are upgrading from a version prior to eZ Publish 3.7.0, you should first upgrade to 3.7.0 as described in this section.

Please make sure that you have a working backup of the site before you do the actual upgrade. The upgrade procedure consists of the following steps:

1. Upgrading the distribution files to 3.7.9
2. Upgrading the database to 3.7.9
3. Running the system upgrade scripts
4. Updating INI settings for "html" classification
5. Clearing the caches

Step 1: Upgrading the distribution files

The easiest way to upgrade the distribution files is to unpack eZ Publish 3.7.9 to a directory and then copy the directories that contain site-specific files from the existing installation. Make sure that you copy the following directories:

- design/example
- design/example_admin
- var
- settings/siteaccess
- settings/override

Replace "example" and "example admin" with actual names used by your siteaccesses.

Custom extensions

If you are using custom extensions then the sub-directories inside the "extension" directory will also have to be copied. However, make sure that you do not overwrite any extensions that come with eZ Publish (for example the "PayPal" extension).

Step 2: Upgrading the database

To upgrade 3.7.0 database to 3.7.9, you should navigate into the eZ Publish 3.7.9 directory and run the following database upgrade scripts one after another:

1. dbupdate-3.7.0-to-3.7.1.sql
2. dbupdate-3.7.1-to-3.7.2.sql
3. dbupdate-3.7.2-to-3.7.3.sql
4. dbupdate-3.7.3-to-3.7.4.sql
5. dbupdate-3.7.4-to-3.7.5.sql
6. dbupdate-3.7.5-to-3.7.6.sql
7. dbupdate-3.7.6-to-3.7.7.sql
8. dbupdate-3.7.7-to-3.7.8.sql
9. dbupdate-3.7.8-to-3.7.9.sql

MySQL

The database upgrade scripts are located in the "update/database/mysql/3.7/" directory of your eZ Publish installation. Each of these scripts can be launched using the following shell command:

```
mysql -u <username> -p<password> <database> < update/database/mysql/3.7/
dbupdate-3.7.x-to-3.7.y.sql
```

PostgreSQL

The database upgrade scripts are located in the "update/database/postgresql/3.7/" directory of your eZ Publish installation. Each of these scripts can be launched using the following shell command:

```
psql -d <database> -U <dbowner> < update/database/postgresql/3.7/
dbupdate-3.7.x-to-3.7.y.sql
```

Step 3: Running the system upgrade scripts

There are currently no upgrade scripts for upgrading from 3.7.x to 3.7.y.

Step 4: Updating INI settings for "html" classification

In eZ Publish versions from 3.7.3 to 3.7.8 it is possible to include HTML code in XML blocks by using the "html" classification. However, this feature can also be used to insert JavaScript code, making it possible for users with sufficient privileges to make an XSS exploit. This feature is therefore disabled by default in eZ Publish 3.7.9 and later, and should only be enabled if you really trust your editors.

To enable html classification, add the following lines to your "content.ini.append.php" file:

```
[literal]
# The class 'html' is disabled by default because it gives editors the
# possibility to insert html and javascript code in XML blocks.
# Don't enable the 'html' class unless you really trust all users who has
# privileges to edit objects containing XML blocks.
AvailableClasses[] = html
```

Step 5: Clearing the caches

Whenever an eZ Publish solution is upgraded, all caches must be cleared in a proper way. This should be done from within a system shell:

1. Navigate into the eZ Publish 3.7.9 directory.
2. Run the clear cache script:

```
bin/shell/clearcache.sh --clear-all
```

Please make sure that all caches are cleared. Sometimes the script is unable to clear caches because of restrictive file/directory permission settings. Make sure that all caches have been cleared by inspecting the contents of the various cache sub-directories within the "var" directory.

1.8 Removing eZ Publish

This section describes how to completely remove an eZ Publish installation from a system. If you have installed eZ Publish using a bundle package, please refer to the "Removing an eZ Publish bundle" (page 87) section.

Removing eZ Publish is done in four steps:

1. Deleting the eZ Publish directory
2. Removing of the database
3. Reconfiguring Apache (optional)
4. Removing the cron job (optional)

WARNING! By following these steps, you will remove both eZ Publish and all the data/content that you have put into the system. Everything will be lost.

Deleting the eZ Publish directory

Remove the eZ Publish directory using your favorite tool.

Linux/UNIX

On Linux/UNIX systems, the removal would most likely be carried out using the "rm" command:

```
$ rm -Rf /path/to/ez_publish
```

Please note that some file/directory permissions might be messed up. If this is the case, it will prevent a regular user from removing all eZ Publish files. You'll probably have to gain root access to solve this problem.

Windows

Windows users may simply delete the eZ Publish directory using the "Explorer".

Removing the database

MySQL

1. Start the MySQL client, log in using your username and password:

```
$ mysql -u <username> -p
```

1

If the username/password is correct, the client will then present a "mysql>" prompt.

2. Delete/remove the database using the drop command followed by the name of the database used by eZ Publish:

```
mysql> drop database <database-name>;
```

PostgreSQL

1. Remove the database by executing the PostgreSQL dropdb command from shell:

```
$ dropdb <database-name>
```

Reconfiguring Apache (optional)

If a virtual host setup was used, it is likely that the Apache configuration file contains eZ Publish specific settings. These settings will not be needed anymore and thus they can be removed. Open the "httpd.conf" file using a text editor, scroll down to the bottom and remove the eZ Publish specific virtual host settings. Remember to restart Apache after altering the configuration file.

Removing cron jobs (optional)

Windows users should skip this part. If cron was configured to run eZ Publish specific jobs, then these will have to be removed. You may have to edit a global cron file (under "/etc/cron*") or use the "crontab" command with the -e (edit) parameter to edit a user's private cron file. Remove the eZ Publish specific entries.

1.8.1 Removing an eZ Publish bundle

This section describes how to remove a bundled eZ Publish installation from both Linux based systems and Windows.

Removing an eZ Publish bundle from a Windows system

1. Log in using the "Administrator" account.
2. Make sure there are no applications running.
3. Choose "Add/Remove" programs from the "Control Panel."
4. Select eZ Publish and click "Remove".
The eZ Publish bundle will be automatically removed.

Removing an eZ Publish bundle from a Linux/UNIX based system

Automated removal

1. Become the root user:

```
$ su -
```

2. Start the installation wizard (from where the bundle was unpacked):

```
# cd /path/to/ezpublish-x.x-x.OS-DISTRO/
# ./install.sh
```

3. Choose "Uninstall" from the main menu.
The eZ Publish bundle will be automatically removed.

Manual removal

Use this method if the install wizard fails or if the directory where the eZ Publish bundle was unpacked to has been removed.

1. Become the root user:

```
$ su -
```

2. Stop the services (Apache and MySQL):

```
# /opt/ezpublish/bin/ezpublish stop
```

3. Remove the eZ Publish bundle directory:

```
# rm -fr /opt/ezpublish
```

4. Remove the lock directory (contains the lock file):

```
# rm -fr /var/state/ezpublish
```

your
information

1.9 Extensions

Extensions are plugins to eZ publish, providing additional custom functionality. Various extensions are available for eZ publish. All of them require the same basic steps for an installation. This chapter will show how to perform the following:

1. Extract the compressed archive containing the extension
2. Activate the extension

Some extensions might require further action to make them fully functional, e.g. creating new database tables, adding certain content classes to eZ publish, etc. Such additional measures are explained in the documentation for each extension.

As outlined before, this section deals with the basic steps only. For demonstration purposes, the installation will be exemplified by an imaginary extension called "ezfoo".

1.9.1 Extracting the files

Each extension is distributed as a compressed archive. The name of the archive file includes the name of the extension and its release version. Furthermore, the compression type is indicated by the file ending, either "tgz", "tar.gz", "bz2", or "zip". For example:

- ezfoo-extension-1.0.tgz
- ezfoo-extension-1.0.tar.gz
- ezfoo-extension-1.0.bz2
- ezfoo-extension-1.0.zip

Extension base directory

The archive must be copied into the "extension/" directory in the root of the eZ publish installation. If this directory does not exist yet, then create it. Make sure that you do **not** create the folder with the plural naming "extension s/" - this is a common error.

For example, on a Linux system do

```
# create extension/ directory  
mkdir /path/to/ezpublish/extension/  
  
# copy the archive  
cp /download/dir/ezfoo-extension-1.0.tar.gz /path/to/ezpublish/extension/
```

Unpack the archive

Next, unpack the files from the archive. See the following table for the correct command to chose on a Linux-like system, depending on the compression type:

archive type	command to extract
tar.gz or tgz	<code>tar -zxvf ezfoo-extension-1.0.tar.gz</code> or <code>tar -zxvf ezfoo-extension-1.0.tgz</code>
bz2	<code>tar -jxvf ezfoo-extension-1.0.bz2</code>
zip	<code>unzip ezfoo-extension-1.0.zip</code>

Windows users can extract an archive with common tools like WinZip, which preferably work with the "zip" file.

Extracting the archive will add the directory "ezfoo/" to the extension folder:

```
/path/to/ezpublish/extension/ezfoo/
```

your
information

1.9.2 Activating the extension

Each extension needs to be activated, which means that it is being registered for eZ publish to be available from within the eZ publish framework. Every extension can either be activated in the eZ publish administration interface or in a configuration file. Furthermore, the activation can be done either for the whole eZ publish installation or for only certain siteaccesses.

Administration interface

Go to the administration interface of your eZ publish installation and navigate to "Setup / Extensions". The sample extension should be available in the list by the name "ezfoo". Register this extension by activating the checkbox, then click "Apply changes".

(see figure 1.26)

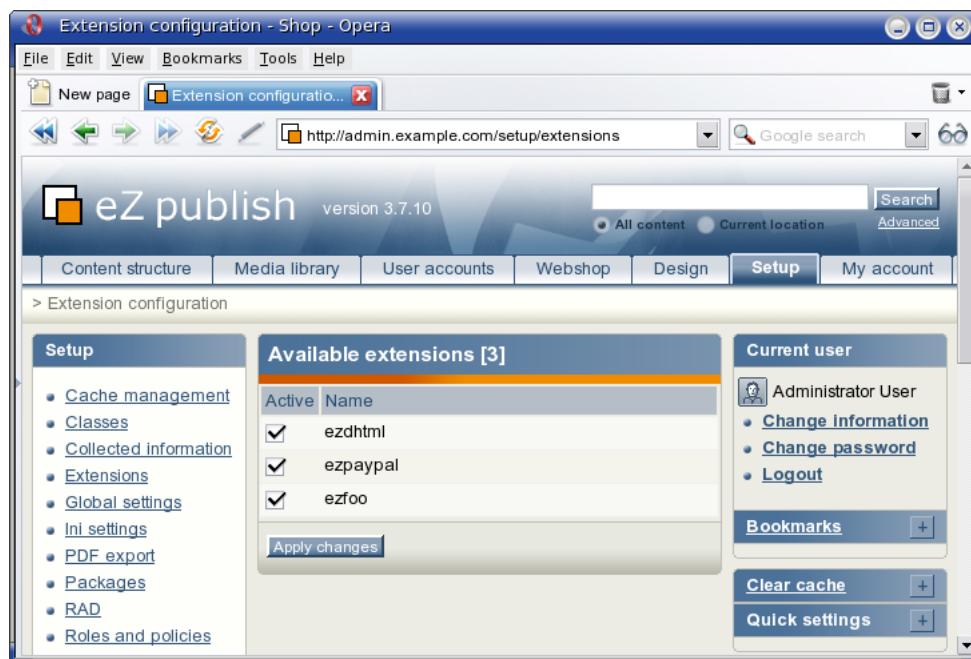


Figure 1.26: Screenshot of extension configuration in administration interface.

This will activate the extension for all siteaccesses of an eZ publish installation.

Configuration file

Alternatively, an extension can be enabled manually in the site.ini (page 1206) configuration file.

Activating for the whole installation

To do so, edit "site.ini.append[.php]" in the folder

```
/root_of_ezpublish/settings/override/
```

If this file does not exist, create it. Locate (or add) the configuration block (page 140) "[ExtensionSettings]" and add the line which is registering the extension:

```
[ExtensionSettings]
ActiveExtensions[] =ezfoo
```

Multiple extensions can be present within the "[ExtensionSettings]" block.

Activating for certain siteaccesses

If you run several virtual sites aka siteaccesses based on a single eZ publish installation and only some of the sites should use the extension, then make the changes in the override file of that siteaccess. This file could be located e.g. at

```
/root_of_ezpublish/settings/siteaccess/news/site.ini.append[.php]
```

Notice that the line registering the extension is not called "ActiveExtensions", but "ActiveAccessExtensions":

```
[ExtensionSettings]
ActiveAccessExtensions[] =ezfoo
```

1.10 Troubleshooting

This section will explain what can be done if installation fails because of some unknown reason.

First of all, make sure that all the requirements without exception are met. The requirements are strict and extremely important. Please read them very carefully. (We usually receive a lot of questions from people who try to use eZ publish 3 with PHP 5.x, Apache 2.x for Windows etc although the requirements do not allow this.)

If all the requirements are met but you still have problems, it is recommended to check the debug information during the installation process. To enable the debug output, do the following:

1. Go to the "settings/override" directory of your eZ publish installation.
2. Create a new file called "site.ini.append.php" and put the following lines to it:

```
[DebugSettings]
DebugOutput=enabled
```

The debug output will appear at the bottom of the page as shown in the following screenshot.

(*see figure 1.27*)

The debug output will be displayed in the setup wizard, in the administration interface and on the actual site. This option can be disabled at any time by replacing "enabled" with "disabled" in the same place of the configuration file.

Please note that the "CheckValidity (page [1364](#))" setting located in the "[SiteAccessSettings]" section of the same file controls if the setup wizard should automatically start the first time the site is accessed/browsed. If you want to restart the wizard after its successful finishing, you can specify "CheckValidity=true" in the "settings/override/site.ini.append.php" file so that the setup wizard will be initiated when trying to access the site.

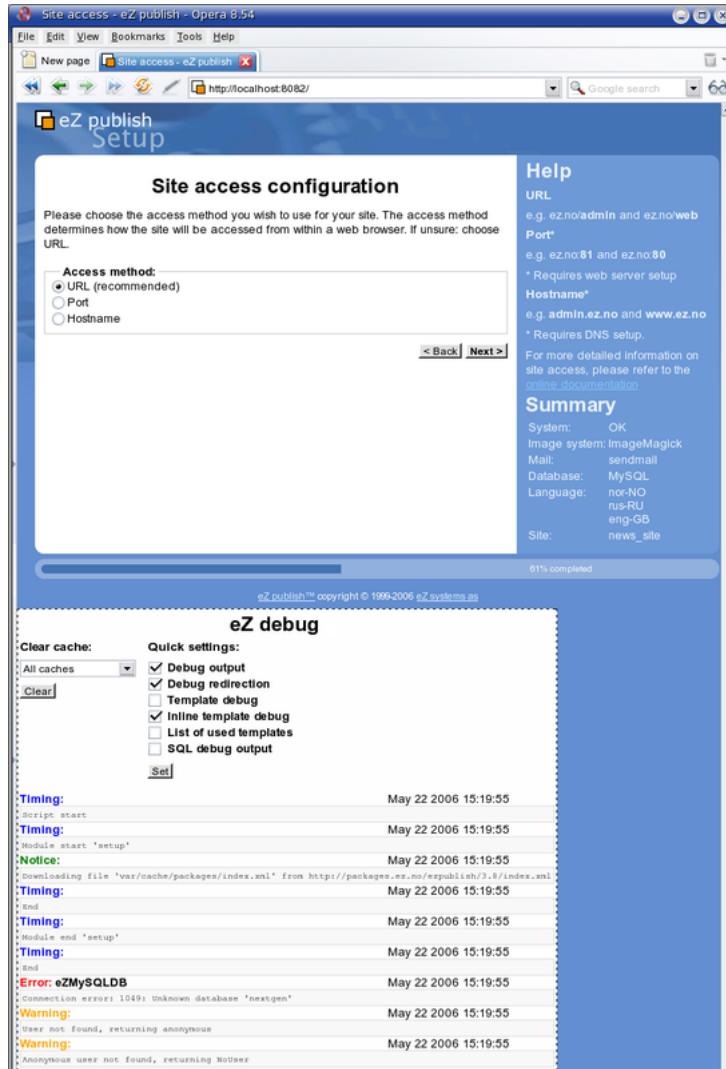


Figure 1.27: *The debug output appears at the bottom of the page*

Chapter 2

Concepts and basics

The purpose of this chapter is to introduce and describe the most important concepts of eZ publish. A rookie developer should definitively read through this chapter in order to understand the basic terms, models, structures and building blocks of the system. This chapter is more generic than technical, it is meant to teach the concepts rather than explaining details. People previously unfamiliar with eZ publish should be able to collect enough information in order to understand the following issues:

- The way eZ publish is built up
- The main directory structure
- The concept and necessity of separating content and design
- How eZ publish stores and manages content
- How eZ publish handles issues related to design
- How eZ publish manages different sites
- The concept of modules and views
- The way eZ publish works with URLs
- The configuration system
- The structure of the workflow system
- How the access/permission system works
- How the webshop works
- A typical page request cycle

2.1 Designs

This section explains the concept of designs and how eZ publish handles different designs. As mentioned in the beginning of this chapter, design is all about the way actual content is marked up and visually presented. When talking about a design, we're talking about the things that make up a web interface: HTML, style sheets, images that are not a part of the content, etc. All files that are related to appearance reside in the "design" directory. An eZ publish installation is capable of handling a virtually unlimited number of designs. Each design has its own dedicated subdirectory within the main design directory. The name of a subdirectory also functions as the actual name of a design. A typical eZ publish design consists of the following components:

- CSS files
- Image files
- Font files
- Template files

Among other things, a siteaccess dictates which design that should be used. By making use of different siteaccesses, it is possible to combine different content and designs. A typical eZ publish site consists of two siteaccesses: a public interface for visitors and a restricted interface for administrators. Both siteaccesses use the same content (database and var directory) but they use different designs. In particular, the administration siteaccess would most likely use the built in administration design. The public siteaccess would use a custom design.

Default designs

An eZ publish distribution comes with at least two default designs:

- admin
- standard

The "admin" directory contains all design related files that make up the built in administration interface. The "standard" directory contains a set of standard/default design related files such as the default/standard templates, images, etc. The contents of these directories should not be tampered with. Instead, custom designs should be used (if/when necessary). A custom design can be added by creating a new subdirectory within the main "/design" directory.

Design directory structure

All files that belong to a specific design are located inside the directory of that design. The name of the directory also functions as the name for the design itself. A eZ publish design directory typically contains the following subdirectories:

Subdirectory	Description
fonts	Font files used by the "texttoimage" (page 832) template operator which is capable of visualizing text using truetype fonts.
images	Non-content specific images (banners, logos, graphical layout elements, etc.).
override	Custom templates that will be used by instead of the default/standard templates. These files will be triggered by template override rules that are specified in a configuration override for "override.ini". Please refer to "The template override system" (page 222) section of the "Templates" chapter for more information about this feature.
stylesheets	CSS files.
templates	Main template(s) (for example the pagelayout, header, footer, etc.) and custom templates that will be used instead of the standard/default templates.

2.1.1 Design combinations

A siteaccess may make use of several designs. This means that the final result generated by eZ publish (the actual HTML) can be a combination of files originating from various designs. A siteaccess is capable of using a combination of the following:

- One main design
- None or several additional designs
- One standard design

A siteaccess should always have at least a main design and a standard design. While the main design can be set to anything, the standard design should not be modified. The default configuration is to use the built-in standard design. It ensures that eZ publish always finds the necessary templates and thus any kind of content can be rendered without problems. A more in-depth explanation is presented below.

Automatic fallback

If eZ publish is unable to find a design specific file (a stylesheet, a template, an image, etc.) within the main design, it will automatically attempt to locate the file elsewhere. The system will sequentially go through all the additional designs (if specified), looking for the requested file. At last, if the requested file still hasn't been found, eZ publish will attempt to locate the missing file within the standard design. The following diagram illustrates this functionality.

(see figure 2.1)

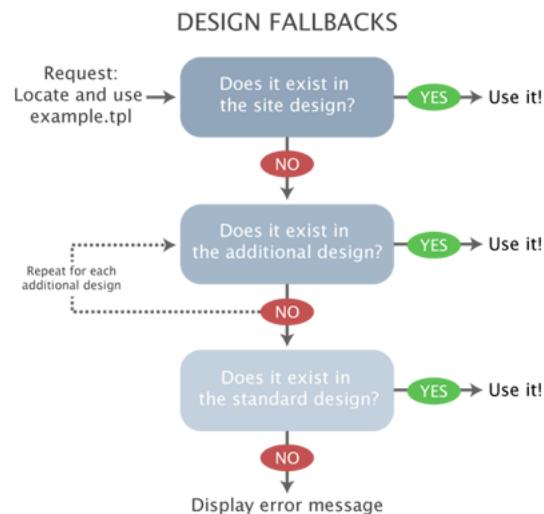


Figure 2.1: *The design fallback mechanism.*

Configuration

The different designs to be used by must be defined in the "[DesignSettings]" block within an override for the "site.ini" configuration file. The following directives can be used:

- SiteDesign
- AdditionalSiteDesignList
- StandardDesign

The "SiteDesign" directive specifies the main design. The "AdditionalSiteDesignList" directive specifies an array of additional site designs. The "StandardDesign" directive specifies the standard design. Even though it is possible to change the standard fallback design, it is not a good idea to do so. The "StandardDesign" directive should always be set to the built-in standard design. This is already defined in the default "site.ini" file and thus there is no need to set the standard design from within an override. If there is a need for a custom fallback design, it should be specified using the "AdditionalSiteDesignList" setting. The automatic fallback mechanism opens up for a lot of possibilities and flexibility. For example, it makes the reuse and combination of designs an easy matter.

Example

The following example shows how to configure the following design settings in an override for the "site.ini" configuration file:

- "my_design" should be the main design
- "fallback_one" should be the first additional design
- "fallback_two" should be the second additional design
- "standard" should be the standard fallback design

```
...
[DesignSettings]
SiteDesign=my_design
AdditionalSiteDesignList []=fallback_one
AdditionalSiteDesignList []=fallback_two
StandardDesign=standard
...
```

In this particular case, if eZ publish is unable to find the requested file within the main design "my_design", it will automatically fallback to the additional designs. At first, the system will look for the requested file within the "fallback_one" design directory. If the requested file is not found, the system will look in the "fallback_two" design directory. If the file still hasn't been found, the system will attempt to locate it within the "standard" design directory. The standard directory will most likely contain the requested file (unless a custom template/override is requested).

2.2 The internal structure of eZ publish

This section describes the internal structure of eZ publish by presenting an brief overview of the different software-layers of the system. eZ publish is a complex, object oriented application written in the PHP language. The system consists of three major parts:

- Libraries
- Kernel
- Modules

The following illustration shows how the different parts of the system are connected.
(see figure 2.2)

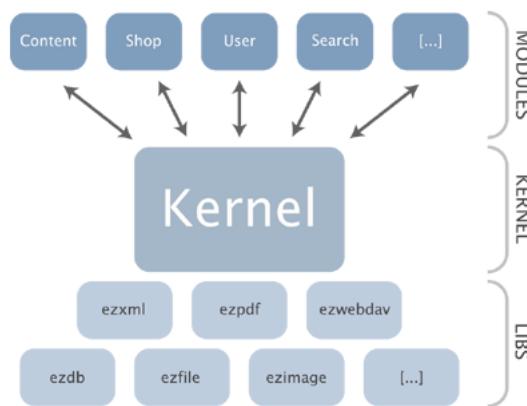


Figure 2.2: Libraries, kernel and modules.

The libraries

The libraries are the main building blocks of the system. These are reuseable general purpose PHP classes. The libraries are in no way dependent on the eZ publish kernel. However, some of them are strongly interconnected and thus inseparable. People looking for general PHP libraries should take a look in the "lib" folder within the root directory of an eZ publish installation. The reference chapter contains a complete list and a short description of the currently available libraries (page [1449](#)).

The kernel

The eZ publish kernel can be described as the system core. It takes care of all the low level functionality like content handling, content versioning, access control, workflows, etc. The kernel consists of various engines that build upon and make use of the general purpose libraries.

The modules

An eZ publish module offers an HTTP interface which can be used for web based interaction with the system. While some modules offer an interface to kernel functionality, others are more or less independent of the kernel. eZ publish comes with a collection of modules that cover the needs of typical everyday tasks. For example, the content module provides an interface that makes it possible to use a web browser to manage content. The reference chapter contains a complete list and a short description of all the currently available modules (page 371). A module can be broken down into the following components:

- Views
- Fetch functions

A view provides an actual web interface. For example, the "search" (page 517) view of the "content" (page 407) module provides a web interface to the built-in search engine. Every eZ publish module provides at least one view. A fetch function makes it possible to extract data through a module from within a template. For example, the "current_user" (page 661) fetch function of the "user" (page 658) module makes it possible to access information related to the user who is currently logged in. Some modules provide fetch functions, some don't.

2.2.1 Directory structure

The eZ publish root directory contains multiple subdirectories. Each subdirectory is dedicated to a specific part of the system and contains a collection of logically related files. The following table gives an overview of the main eZ publish directories.

Directory	Description
bin	The "bin" directory contains various PHP, Perl and shell scripts. For example, it contains the "clearcache" script which can be used to clear all eZ publish caches from within a system shell. The scripts are mainly used for manual maintenance.
cronjobs	The "cronjobs" directory contains miscellaneous scripts for automated periodical maintenance.
design	The "design" directory contains all design related files such as templates, images, stylesheets, etc.
doc	The "doc" directory contains documentation and change logs.
extension	The "extension" directory contains eZ publish plugins. The extension system of eZ publish allows external code to plug in and co-exist with the rest of the system. By using extensions it is possible to create new modules, datatypes, template operators, workflow events and so on.
kernel	The "kernel" directory contains all the kernel files such as the core kernel classes, modules, views, datatypes, etc. This is where the core of the system resides. Only experts should tamper with this part.
lib	The "lib" directory contains the general purpose libraries. These libraries are collections of classes that perform various low level tasks. The kernel makes use of these libraries.
packages	The "packages" directory contains the bundled packages (themes, classes, templates, etc.) that can be installed using either the setup wizard or the administration interface.
settings	The "settings" directory contains dynamic, site specific configuration files.
share	The "share" directory contains static configuration files such as codepages, locale descrip-

	tions, translations, icons, etc.
support	The "support" directory contains the source code for additional applications that can be used to do various advanced tasks. For example, it contains the "lupdate" program that can be used to create and maintain eZ the translation files.
update	The "update" directory contains various scripts that should be used when an eZ publish installation is being upgraded.
var	The "var" directory contains cache files and logs. It also contains actual content that doesn't go into the database (images and files). The size of this directory will most likely increase as the system is being used.

2.3 Content and design

This section explains the fundamental concepts of content and design. It is important to understand what content and design actually are, how they interconnect and how the system handles these fundamental elements.

Content

In the world of eZ publish, content and design are separated. By content we mean information that is to be organized and stored using some structure. For example, it may be the actual contents of a news article (title, intro, body, images), the properties of a car (make, model, year, color) and so on. In other words, all custom information that is stored for the purpose of later retrieval is referred to as content.

Design

The information stored in a content structure must be presented somehow, preferably in a way that is easily understood by humans. While content means actual data, design is all about the way the data is marked up and visually presented. When talking about design, we're talking about the things that make up a web interface: HTML, style sheets, images that are not a part of the content, etc.

Templates

eZ publish uses templates as the fundamental unit of site design. For example, a template might dictate that a page should appear with the site's title bar on the top, and then main content in the middle. When the page is accessed, it then becomes the content management system's job to "flow" the content into the appropriate places in the template. An eZ publish template is basically a custom HTML file that describes how some particular type of content should be visualized. In addition to standard HTML syntax, it is possible to use eZ publish specific code to for example extract content from the system. The HTML syntax in the built-in/default templates follow the XHTML 1.0 Transitional specification.

The separation of content and design

While content is all about storing and structuring custom/raw data, the purpose of the design is to dictate how the content should be visualized. The result of a combination of these elements is a complete interface, as illustrated in the following diagram.

(see figure 2.3)

This distinction, and the system's ability to handle it is one of the key features of eZ publish. The separation of content and design opens up an entire range of possibilities that simply cannot

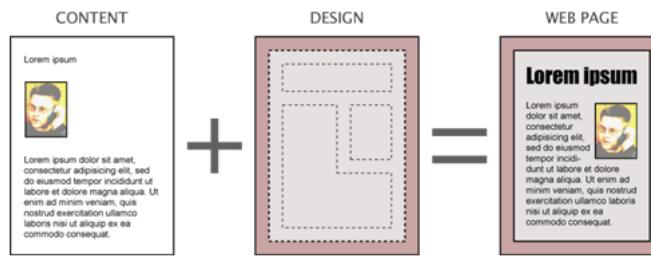


Figure 2.3: *Content + Design = Web page*

be achieved otherwise. The following list outlines some of the most important benefits of this technique:

- Content authors and designers can work separately without conflicts
- Content can be published easily in multiple formats
- Content can easily be transferred and re-purposed
- Global redesigns/changes can be applied by simple modifications

2.3.1 Storage

This section explains where eZ publish stores information that belongs to a site (not the system itself). A typical eZ publish site consists of the following elements:

- Actual content
- Design related files
- Configuration files

Actual content is structured and stored inside a database. This is true for all content except for images and files, which are stored on the filesystem. The main reason for this is because the filesystem is much faster than the database when it comes to the storage and retrieval of large data chunks. Having the files on the filesystem allows the webserver to serve them directly without the need of going through the database. In addition, this technique makes it easier to use external tools to manipulate/scan/index the contents of the uploaded files. For example, the built in search engine is capable of using external utilities to index the contents of miscellaneous files (PDF, Word documents, Excel sheets, etc.). Having the files on the filesystem dramatically decreases the size of the database and thus makes it easier to copy and handle. Everything that is related to design (template files, CSS files, non content specific images, etc.) and configuration settings are also stored on the filesystem. A backup of an eZ publish site must therefore contain both a dump of the database and a copy of the necessary files. The following illustration shows an overview of how the system makes use of the database and the filesystem to store the different elements of a site.

(see figure 2.4)

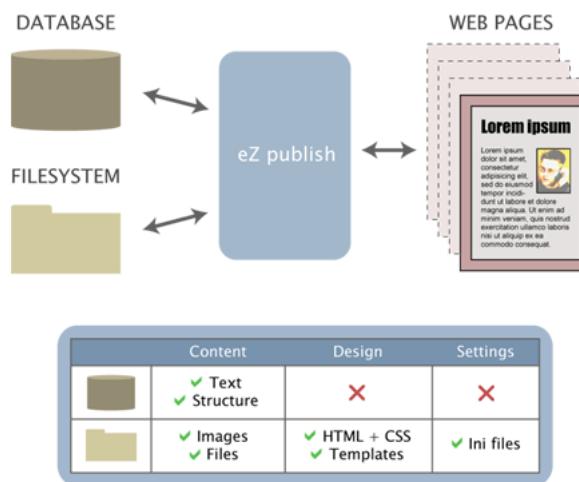


Figure 2.4: Storage overview

2.4 Content management

The role of a content management system is to organize and store content regardless of type and complexity. The main goal of such a system is to provide a well structured, automated yet flexible solution allowing information to be freely distributed and instantly updated across various communication channels (such as the world wide web, intranets and miscellaneous front and back-end systems). This section describes how eZ publish actually handles content.

A typical example

Let's consider a scenario at a university with a need of storing information about students. Most off-the-shelf content management systems will offer a selection of built in content types. There might for example exist a "Person" type, consisting of fields like "name", "birthdate", "phone number" and so on. However, the custom student data will probably not fit perfectly into this predefined model since it might consist of information that is specific for the university (for example student ID, department, etc.). Even though some systems allow the creation of custom structures, the solution is often a complicated and timeconsuming process that requires both programming and manipulation of the database. In addition, once the solution is in use, future alteration of the structure itself will most likely become a problem.

Content management in eZ publish

Unlike other content management systems, eZ publish does not make use of a predefined "one-size-fits-all" approach. Instead of desperately trying to fit data into predefined and rigid structures, the system allows the creation of custom structures by the way of a unique object oriented approach. For example, the site developer can build custom structures that perfectly satisfies the storage needs of the university. This is one of the key features that make eZ publish a flexible and successful system. In addition to offering the freedom of custom structures, it also allows the modification of the content structures at runtime. In other words, if the custom student structure used in the example above needs to be modified, then eZ publish will automatically alter it based on the administrator's commands.

Although the possibility to create and modify content structures is a wonderful feature, there isn't always need for using it. This is why an eZ publish distribution comes with a selection of predefined content structures and thus allows the developer to choose between the following scenarios:

- Use the standard/built-in structures
- Use modified versions of the standard/built-in structures
- Use only custom structures
- Use a combination of standard, modified and custom structures

An object oriented content structure

The eZ publish content structure is based on ideas borrowed from the object oriented world of popular programming languages like Smalltalk, C++, JAVA, etc. Superficially, object-oriented means nothing more than looking at the world in terms of objects. In real life, people are surrounded by several objects: furniture, cars, pets, humans, etc. Each of these objects have traits that we use to identify them. This is also the way eZ publish defines and manages content.

The system offers a selection of fundamental building blocks and mechanisms that together provide a flexible content management solution. An actual data structure is described using something called a *content class*. A content class is built up of attributes. An *attribute* can be thought of as a field, for example the "birthdate" field in a structure designed to store information about students. The description of the entire structure would be referred to as the "student class". The characteristics of an attribute inside the class are determined by the *datatype* that was chosen to represent that attribute.

It is important to understand that a content class is just a definition of an arbitrary structure. In other words, the class itself describes the structure but it does not store any actual data. Once a content class has been defined, it is possible to create instances of that class. An instance of a content class is called a *content object*. Actual content is stored inside objects of different types. A content object consists of one or more *versions*. The versioning layer makes it possible to have different versions of the same content. Each version consists of one or more *translations*. The translation layer makes it possible to represent the same version of the same content in multiple languages. A translation consists of *attributes*. The attributes are the final elements in the content structure chain, this is where actual data is stored.

The content objects are wrapped and organized by the way of *nodes* that are placed inside a tree-like structure. This tree is often referred to as the *node tree*. The following sections contain comprehensive explanations related to the elements that were introduced above.

2.4.1 Datatypes

A datatype is the smallest possible entity of storage. It determines how a specific type of information should be validated, stored, retrieved, formatted and so on. eZ publish comes with a collection of fundamental datatypes that can be used to build powerful and complex content structures. In addition, it is possible to extend the system by creating custom datatypes for special needs. Custom datatypes have to be programmed in PHP. However, the built in datatypes are usually sufficient enough for typical scenarios. The following table gives an overview of the most basic datatypes that come with eZ publish.

Datatype	Description
Text line (page 321)	Stores a single line of unformatted text
Text block (page 319)	Stores multiple lines of unformatted text
XML block (page 329)	Validates and stores multiple lines of formatted text
Integer (page 293)	Validates and stores a numerical integer value
Float (page 284)	Validates and stores a numerical floating point value

Please refer to the "Datatypes" (page 268) section of the reference chapter for a comprehensive list of all the built-in datatypes. Additional datatypes can be downloaded from <http://ez.no/community/contribs/datatypes>; they are created by the members of the eZ publish community.

Input validation

As the list above indicates, some datatypes take care of more than just storing data. For example, the "XML block" datatype apparently supports validation. This means that the inputted XML will be validated before it is actually stored in the database. In other words, the system will only accept and store the data if it is a valid XML structure. Input validation is supported by most (but not all) of the built in datatypes. The validation feature of a datatype can not be turned on or off. In other words, if a datatype happens to support validation, it will always try to validate the incoming data and thus the system will never allow the storage of incorrectly formatted input.

2.4.2 The content class

A content class is a definition of an arbitrary data structure. It does not store any actual data. A content class is made up of attributes. The characteristics of an attribute are determined by the datatype that is chosen for that specific attribute. By combining different datatypes, it is possible to represent complex data structures. The following illustration shows the anatomy of a content class called "Article", which defines a data structure for storing news articles. It consists of attributes dedicated for storing the title, an introduction text and the actual body of an article.

(see figure 2.5)

ARTICLE CLASS		
ATTRIBUTES	Name	Datatype
	Title	Text line
	Intro	Text line
	Body	XML field

Figure 2.5: Example of a content class.

An eZ publish distribution comes with a set of general purpose classes (page 344) that are designed for typical web scenarios. For example, the default image class defines a structure for storing image files. It consists of attributes for storing the name of the image, the actual image file, the caption and an alternative image text. The built-in classes can be modified in order to become more suitable for a specific case. In addition, it is possible to create completely new and custom classes. Content classes can be created, modified and removed easily using the administration interface. When a content class is removed, all instances of that class (containing actual data) will also be removed from the system. The following screenshot shows the class edit interface in action.

(see figure 2.6)

Class structure

A content class consists of the following elements:

- Name
- Identifier
- Object name pattern
- Container flag
- Attributes

The screenshot shows the 'Edit <Documentation page> [Class]' interface. At the top, it says 'Last modified: 28/01/2005 3:00 pm, Balazs Halasy'. The 'Name:' field contains 'Documentation page'. The 'Identifier:' field contains 'documentation_page'. The 'Object name pattern:' field contains '<title>'. The 'Container:' field has a checked checkbox. Below this is a list item '1. Title [Text line] (id:183)' with a delete button. The 'Name:' field for this item is 'Title' and the 'Identifier:' field is 'title'. At the bottom, there are checkboxes for 'Required' (checked), 'Searchable' (checked), 'Information collector' (unchecked), and 'Disable translation' (unchecked). There is also a 'Default value:' field.

Figure 2.6: The class edit interface.

Name

The name is for storing a user friendly name which describes the data structure that the class defines. A class name can consist of letters, digits, spaces and special characters. The maximum length is 255 characters. For example, if a class defines a data structure for storing information about graduate students, the name of the class would most likely be "Graduate student". This name will appear in various class lists throughout the administration interface, but it will not be used internally by the system. If a blank name is provided, eZ publish will automatically generate a unique name when the class definition is stored.

Identifier

The identifier is for internal use. In particular, class identifiers are used in configuration files, templates and in PHP code. A class identifier can only consist of lowercase letters, digits and underscores. The maximum length is 50 characters. For example, if a class defines a data structure for storing information about graduate students, the identifier of the class would probably be "graduate_student". If a blank identifier is provided, eZ publish will automatically generate a unique identifier when the class definition is stored.

Object name pattern

The object name pattern controls how the name of an actual object (an instance of a class) will be generated. A pattern usually consists of attribute identifiers (described later) that tell

eZ publish about which attributes it should use when generating the name of an object. Each attribute identifier has to be encapsulated by angle brackets. Text outside the angle brackets will be included directly. If a blank pattern is provided, eZ publish will automatically use the identifier of the first attribute.

Container flag

The container flag controls whether an instance of the class should be allowed to have sub items (often called child nodes, children) or not. This setting only affects the administration interface, it was added in order to provide a more convenient environment for administrators and content authors. In other words, it doesn't control any actual low level logic, it simply controls the way the graphical user interface behaves.

Attributes

As pointed out earlier, it is the structure and type of the attributes that make up the actual data structure that the class defines. A content class must at least have one attribute. On the other hand, a class can contain virtually an unlimited number of attributes. Class attributes can be added, removed and rearranged at any time using the administration interface. If an attribute is added to a class, it will be added to all current and upcoming instances of that class. If an attribute is removed, it will also be removed from all instances.

Although it is possible to remove and add attributes using the administration interface, in some cases these operations may corrupt the database. This usually happens when there are too many instances that need to be updated. If the required processing time exceeds the maximum execution time for PHP scripts, the sequence will be interrupted and thus the database will most likely be left in an inconsistent state. At the time of writing, this problem can only be solved by increasing the maximum execution time, which is defined in "php.ini" as "max_execution_time". The default value is 30 seconds, it should be increased to a couple of minutes. A more reliable solution (a PHP script that takes care of adding/removing attributes and run it from within a shell) will probably be added in the future.

2.4.3 Class attributes

A content class is made up of one or more attributes where each attribute is represented by a datatype. The characteristics of an attribute are determined by the datatype that is chosen for that specific attribute. An attribute is made up of the following elements:

- Name
- Identifier
- Generic controls
- Datatype specific controls

Name

The name is for storing a user friendly name for the attribute. For example, if the attribute is supposed to store birthdates, the name of the attribute would most likely be "Date of birth". This string will appear in various parts of the administration interface, but it will not be used internally by the system. The name of an attribute can consist of letters, digits, spaces and special characters. The maximum length is 255 characters. If a blank name is provided, eZ publish will automatically generate a unique name for the attribute when the class definition is stored.

Identifier

The identifier of an attribute is for internal use. In particular, attribute identifiers are used in configuration files, templates and in PHP code. An attribute identifier can only consist of lowercase letters, digits and underscores. The maximum length is 50 characters. For example, if the attribute is supposed to store birthdates, the identifier of the attribute would probably be "date_of_birth". If a blank identifier is provided, eZ publish will automatically generate a unique identifier when the class definition is stored.

Generic controls

Each attribute has a set of generic controls. These controls are the same for each attribute, regardless (but not independent) of the datatype that represents the attribute. The generic controls are a set of switches that can be turned on or off:

- Required
- Searchable
- Information collector
- Translatable

Required

The required switch controls the behavior of the storage procedure for content objects (instances of a content class). It can be used regardless of the datatype that represents the attribute. When the required flag of an attribute is set, the system will keep rejecting the inputted data until all required information is provided. If the required flag is unset, eZ publish will not care whether any actual data was provided or not. When an attribute is added, the required switch is off. Please note that inputted data will be validated according to the chosen datatype's validation rules regardless of the state of the attribute's required switch. Input validation is supported by most (but not all) of the built in datatypes. The following example demonstrates how these features actually work.

Let's say that we have created a content class that defines a data structure for storing information about prisoners. The class would typically consist of various attributes for storing different kinds of data: name, identification number, date of birth, cell, block, etc. Having at least the name and the birthdate attributes required will eliminate the possibility of storing convicts without names and/or birthdates. If the birthdate attribute is represented by the built-in "date" datatype, the system will only accept the input if the birthdate is provided using a correct date format.

Searchable

The searchable switch can be used to control whether the actual data stored using the attribute should be indexed by the search engine or if it should be left unindexed. Search indexing is supported by the majority of the built-in datatypes. Please refer to the "Datatypes" (page 268) section of the reference chapter to see which datatypes that support search indexing.

Information collector

The information collector switch can be used to control the attribute's behavior in view mode. The default view mode behavior results in the display of the information that was provided in edit mode. For example, when viewing a news article, the contents of the article are displayed but can not be edited. However, if an attribute is marked as a collector, it will allow information to be input in view mode. At first, this feature might seem a bit odd. However, it is actually quite handy. For example, it can be used to quickly create simple feedback forms. The contents of a form created using this technique will be e-mailed to the site administrator (or to a specified address) once the form is submitted. Information collection is only supported by a small set of the built in datatypes. The following example demonstrates how this feature could be used to create a basic feedback form.

Let's say that we have created a content class called "Feedback form" using the following attributes: name, subject and message. The subject and the message attributes would be marked as information collectors. When an instance of this class is viewed, the subject and the message attributes will be displayed as input fields along with a "Send" button.

Translatable

The translatable switch controls whether actual data stored using the attribute should exist in only one language (the default language) or if it should be possible to translate it using the additional languages. The translation mechanism is completely independent of the datatype layer. In other words, this switch can be used regardless of the datatype that was chosen to represent the attribute.

When an attribute is added, the translation switch is "on". Turning it off is typically useful when the attribute is supposed to store non-translatable input. For example, translating dates, numerical values, prices, email addresses, etc. doesn't make much sense.

Datatype specific controls

An attribute can have a set of additional controls that are specific for the datatype that was chosen to represent that attribute. Some datatypes allow fine grained customization, some not. For example, the built-in "Text line" datatype provides two settings: default value and maximum length.

2.4.4 Object versioning

eZ publish comes with a built in versioning system which is implemented at the object level. This mechanism makes it possible to have several versions of the contents (attributes) of an object. It basically provides a generic, out-of-the-box version control framework that can be used with any kind of content. The different versions are encapsulated by the object itself. The following illustration shows a more detailed example of the object structure seen from the outside world.

(see figure 2.7)

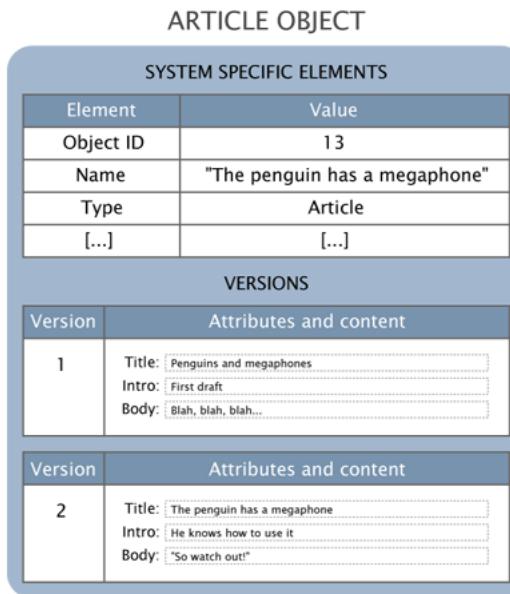


Figure 2.7: Example of a content object that consists of two versions.

Every time an object is edited, a new version of the object's contents will be created. It is always the new version that will be edited, the old version(s) remains untouched. This is how eZ publish keeps track of changes made by various users. An accidental or unwanted change can thus be undone by simply reverting an object back to the previous version.

Version limitations

Since every edit procedure results in the creation of a new version (unless the new version is discarded), the database can be quickly filled up by different versions of the same content. In order to prevent this problem, the versioning system can be limited to a certain number of versions per object. It is possible to assign different version limitations for different object types (different classes). The default limitation is 10, which means that every object can have a maximum number of 10 versions of its content. If the maximum count is reached, the oldest version will be automatically deleted and thus a free slot will be available for the new one. This is the default behavior. An alternative setting can be used to disallow the creation of new versions until an

existing version is manually deleted by a user.

Version structure

A version consist of the following elements:

- Version number
- Creation time
- Modification time
- Creator
- Status
- Translations

Version number

Every version has a unique version number. This number is used by the system to organize and keep track of the different versions of an object. The version number is automatically increased for each version that is created inside an object.

Creation time

The creation time contains a timestamp pinpointing the exact date and time when the version was initially created. This information is set by the system and will remain the same regardless of what happens to the version.

Modification time

The modification time contains a timestamp revealing the exact date and time when the version was last modified. This information is set by the system every time the version is stored and when the version is finally published. When a version is published, the modification time of the object itself will be updated (it will simply be set to the same value as modification time of the version that was published).

Creator

The version's creator contains a reference to the user that created the version. Although a content object can only belong to a single user (revealed by the "Owner" field), each version may belong different users. The creator reference is set by the system when the version is created. It can not be manipulated and will not change even if the user who created the version is removed from the system.

Status

The state of a version is determined by its status. There are five possibilities:

- Draft (0)
- Published (1)
- Pending (2)
- Archived (3)
- Rejected (4)

A newly created version is a *draft*. This status will remain until that version becomes *published*. Although an object can have many versions, there can only be one published version (the others are usually drafts and archived versions). The published version can be considered as the "current" version and it is the one that is accessed when the object is viewed. A published version can not become a draft. However, it will become *archived* as soon as another version is published. The following illustration shows how the versioning system actually works.

(see figure 2.8)

The illustration above shows the most common states of a content object. When a new object is created (step 1), eZ publish will also create a new draft version. Because the object has not been published yet, its status is set to draft and the current version is unknown. Storing the draft (steps 2a and 2b) will not change the state of the object. The only thing that will happen is that the contents of the draft will be stored in version 1. If the draft (which is the only existing version) is discarded, the object is completely removed from the system (step 2c). When the draft is published (step 2), both the draft and the object's states will be set to published. In addition, the current version will be set to 1, which reveals the currently published version of the object. When published, the contents of the object can be viewed by others. A published object can be removed/deleted from the system (step 3a). When removed, the object's state will be set to "Archived" and thus it will be in the trash. The object can be recovered from the trash to its previous state. Among other things, this involves the status field being set to "Published" again. When a published object is edited (step 4), the current version (version 1 in this case) will remain untouched and a completely new version will be created. The contents of the new version (version 2 in this case) will be a copy of the contents of the current version. Again, storing the draft (steps 4b and 4c) will not change the state of the object. If the draft is discarded (step 4a), it will be completely removed from the system and thus the object will be in the exact same state as it was in before it was edited. If the newly created and edited draft is published, it will become the current version of the object and thus the previous version (version 1 in this case) will be set to "Archived". Step 5a illustrates what would happen if the object (now with two versions) would be removed.

The pending and the rejected states are used by the collaboration system. When a version is waiting to be approved by an editor, the status is set to *pending*. If the version is approved, it will be automatically published and thus the status will be set to *published*. On the other hand, if a *pending* version is rejected by the editor, the status will be set to *rejected*.

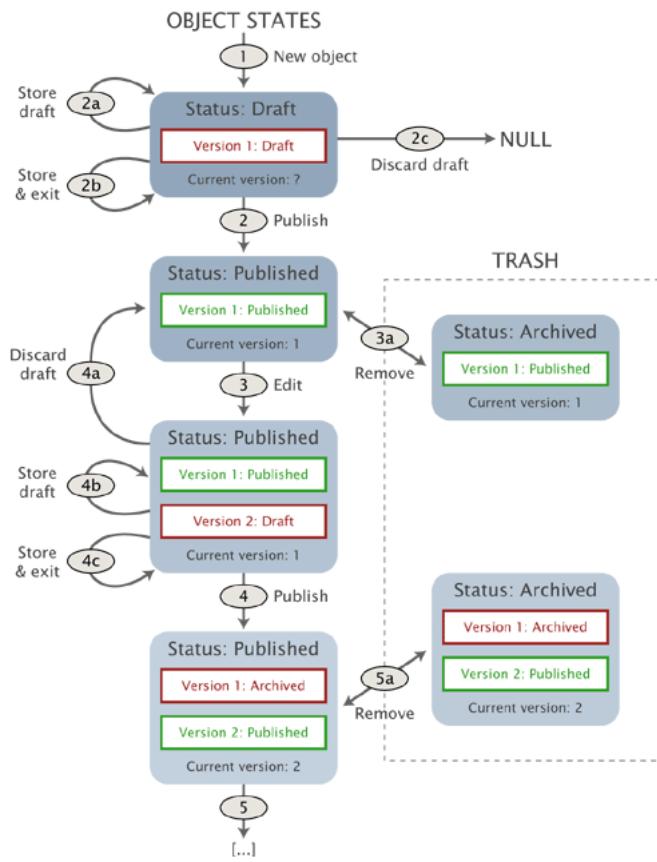


Figure 2.8: Overview of the object states.

A version can only be edited if it is a draft and it can only be edited by the same user who initially created it. In addition, rejected versions can also be edited. When a rejected version is edited, it will become a draft. Published and archived versions can not be edited. However, it is possible to make copies of them. When a published or an archived version is copied, the status of the copy is set to draft and thus it becomes editable. When/if the new draft is published, the system automatically sets the status of the previously published version to archived and the new draft will become the published version.

Translations

The actual contents of a version is stored inside different translations. A translation is a representation of the information in a specific language. In other words, the translation layer allows a version of the object's actual contents to exist in different languages. A version always has at least one translation of the content (which represents the data in the default/standard language).

2.4.5 Multiple languages

In addition to the versioning system, the content model of eZ publish also provides a built-in multilanguage framework. This feature is implemented at the version level and allows a version to exist in several languages. It provides a generic one-to-one translation mechanism that can be used to translate any kind of content. A one-to-one translation solution makes it possible to represent the exact same content in multiple languages. For example, when a news article is available in English, Norwegian and Hungarian (same content in all three cases), we say that we have one-to-one translation of the content. The translation mechanism is completely independent of the datatypes. In other words, any kind of content can be translated regardless of the datatypes that are used to realize the content's structure. It is possible to start with only one language and when time comes, add translations and thus extend the spectrum of the target audience. The following illustration shows an example of an object seen from the outside world. The object has three versions and each version exists in several languages. A language in this case is referred to as a *translation*.

(see figure 2.9)



Figure 2.9: Content object structure (with versions and translations).

As the illustration indicates, each version can have a different set of translations. At the minimum, a version always has one translation, which is the default translation. The default translation of a version can not be removed. However, additional translations can be added and removed when the version is being edited. A translation for a specific language can only be added if it exists in the global content translation list. This list simply keeps track of the languages that users are allowed to use when translating content. For example, if the global translation list contains English as the primary language along with Norwegian and Hungarian as additional languages, all versions will exist at least in English. In addition, for each version it would be possible to add or remove both Norwegian and Hungarian translations.

The global translation list can be manipulated at any time through the administration interface. A translation added to the global translation list will immediately become available for use. Removing a language from the global translation list will (upon confirmation) also result in the removal of all translations that use that language.

Non translatable attributes

The data structure defined by a class is built up of attributes where each attribute is represented by a datatype. Among other things, an attribute of a class can be made translatable or not. If an attribute is translatable, the system will allow the translation of its contents when an object of that class is being edited. This is typically convenient when the attribute contains actual text. For example, the written part of a news article can be translated into different languages. However, some attributes are non-translatable by nature. This is typical for images without text, numbers, dates, e-mail addresses and so on. Such attributes can be made non-translatable and thus their contents will simply be copied from the default translation. The copied values can not be edited.

For example, let's say that we need to store information about furniture in multiple languages. We could build a furniture class using the following attributes: name, photo, description, height, width, depth and weight. Allowing the translation of anything else than the description attribute would be pointless since the values stored by the other attributes are the same regardless of the language used to describe the furniture. In other words, the name, photo, height, width, depth and weight would be the same in for example both English and Norwegian. Conversion between different measuring units would have to be done within the template that is used to display the information.

Translation of content

The easiest way to translate content is by using the same user to add the different translations sent in by the translators. This approach is usually less cumbersome than having a bunch of translators using the system because they would have to be carefully coordinated.

When several translators are involved, each translator must work on his or her own version of the content. The reason for this is that a version can only be edited by the same user who initially created it. The original author must first publish a version. Once a published version exists, it can be copied and translated by multiple translators. However, the translators can not work simultaneously because each version also contains the actual translations. The translation work would have to be carried out sequentially. The following example demonstrates how different

translators can work together to create a news article in three languages: English, Norwegian and Hungarian.

1. The original author composes the article using the default language (English).
2. The article is published and thus it becomes visible for other users.
3. The Norwegian translator comes along and edits the article. The system will automatically create a new version which will be a copy of the published version.
4. The Norwegian translator adds the Norwegian translation to the newly created version.
5. The article is published and will be available in two languages: English and Norwegian.
6. The Hungarian translator comes along and edits the article. The system will automatically create a new version which will be a copy of the published version.
7. The Hungarian translator adds the Hungarian translation to the newly created version.
8. The article is published and will be available in three languages: English, Norwegian and Hungarian.

Access control

It is possible to control whether a user (or a group of users) should be able to translate content or not. This policy can be controlled on a class, section and owner basis. However, there is no fine grained mechanism for controlling access to the different languages. This functionality will be added sometime in the near future. Currently, if a user has access to translate the contents of an object then that user can add, remove and use all the available translations within the object. In addition, it is also possible to control access to the global translation list. This makes it possible to allow users other than the site administrator to add and remove translations on a global basis.

2.4.6 The content node

When the system is in use, new content objects are created on the fly. For example, when a news article is composed, a new article object is created. Obviously, the content objects can't just hover around in space, they have to be organized in some way. This is where the *nodes* and the *content node tree* comes in. A content node is nothing more than an encapsulation of a content object. In eZ publish, every object is usually represented by one or more nodes. The following illustration shows a simplified example of a node and a corresponding object (which is referenced by the node) as it would have been represented inside the system.

(see figure 2.10)



Figure 2.10: *Object - node relation*

The content node tree is built up of nodes. A node is simply a location of an object within the tree structure. The tree is the actual mechanism used to hierarchically organize the objects that are present on the system. The content node tree is explained in the next section.

Node structure

A content node consists of the following elements:

- Node ID
- Parent node ID
- Object ID
- Sort method
- Sort order
- Priority

Node ID

Every node has a unique identification number. The ID numbers are used by the system to organize and keep track of the different nodes. These ID numbers are not recycled. In other words, if a node is deleted, the ID number of that node will not be reused when a new node is created.

Parent node ID

The parent node ID of a node reveals the node's superior node in the tree.

Object ID

Every object that exists in the system has a unique identification number. The object ID of a node pinpoints the actual object that the node encapsulates.

Sort method

The sorting method of a node determines how the children of the node should be sorted. The following sorting methods are possible:

Method	ID	Description
Class identifier	6	The nodes are sorted by the class identifiers of the objects.
Class name	7	The nodes are sorted by the class names of the objects.
Depth	5	The nodes are sorted by their depth in the tree. A node further down in the tree has a higher level of depth. The root node has a depth of 1.
Modified	3	The nodes are sorted by the modification time of the objects.
Modified subnode	10	The nodes are sorted based on the modification time of their children.
Name	9	The nodes are sorted by the names of the objects.
Path	1	The nodes are sorted by their path strings.
Priority	8	The nodes are sorted by their priority. Every node has a priority field that can be set by the user. This solution allows the nodes to be sorted in a custom order. The priority field is described below.
Published	2	The nodes are sorted by the creation time of the objects'

Section	4	current/published versions. The nodes are sorted by the section IDs of the objects.
---------	---	--

Please note that it is possible to combine the available sort methods in order to sort nodes in a more complex way. However, since a node is incapable of "remembering" a combination (you can only set one method and one order for each node), this has to be done in the templates.

Sort order

The sorting order determines the order in which the children of the node should be sorted. There are two possibilities:

- Descending (0 / FALSE)
- Ascending (1 / TRUE)

For example, if the sorting method is set to "Name" and the sort method is set to "Ascending", the underlying nodes will be alphabetically sorted from A to Z. If the sort method is set to "Descending", the underlying nodes will be sorted from Z to A.

Priority

The priority field allows a user to assign both positive and negative integer values to a node (zero is also allowed). This field makes it possible to sort nodes in a custom way. If the sorting method of a node is set to "Priority", the children of that node will be sorted by their priorities.

2.4.7 The content node tree

The content node tree is a hierarchical organization of the objects. Each leaf in the tree is a node (also known as a *location*). Each node refers to one object. The usual case is that an object is referenced by only one node. Because of the node-encapsulation of objects, any type of content object can be placed anywhere in the tree. At the minimum, the tree consists of one node, called the *root node*. The identification number of the root node is 1. The root node is a virtual node, it does not encapsulate an actual object. A node that is directly below the root node is called a *top level node* (the top level nodes are described in the next section). The depth and width of the tree is virtually unlimited. The following illustration shows a simplified example of how objects are referenced by nodes which together make up the content node tree.

(see figure 2.11)

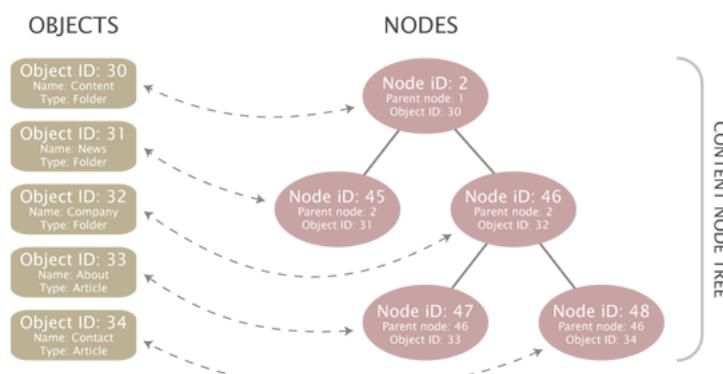


Figure 2.11: Objects, nodes and the content node tree

The following illustration shows the same node structure seen from the outside world.

(see figure 2.12)

Multiple locations

An object may be referenced by several nodes, which means that the same object can appear at different locations within the tree. This feature can for example be used to place a specific news article at two locations: the frontpage and the archive. In the case of multiple nodes/locations, only one node can be considered as the *main node* of an object. The main node usually represents the object's original location in the tree. The other nodes can be thought of as additional nodes / locations. If an object is referenced by a single node then of course that node would be the main node. Among other things, the main node is used to avoid multiple search hits, infinite recursive loops, smart filtering, etc. The following illustration shows an example of a structure where an object has multiple locations in the tree. It will simply be empty and will have the possibility to contain a different set of sub items.

(see figure 2.13)

CONTENT NODE TREE

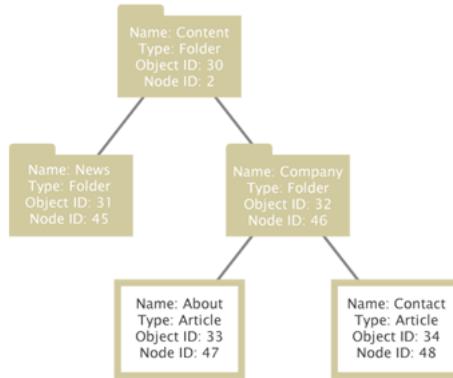


Figure 2.12: *Content node tree*

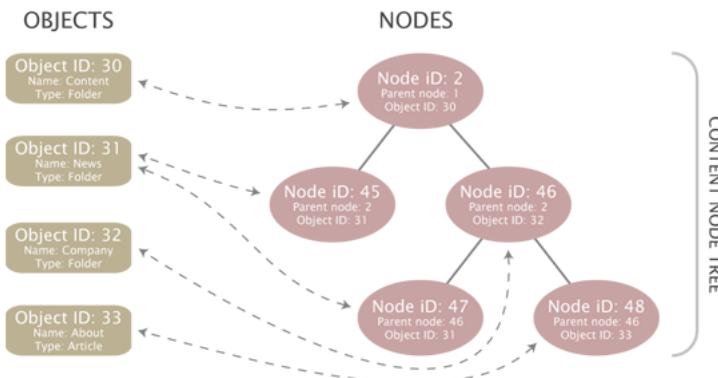


Figure 2.13: *Objects, node and the content node tree - multiple locations*

The following illustration shows the same node structure seen from the outside world.

(see figure 2.14)

Pitfall

A very common mistake when planning the structure of a site is thinking of multiple locations as shortcuts/links on a filesystem. Unfortunately, this is not how the node tree works. When a new location is added to an object, eZ publish will not go through and create replica of the node structure below the object's original location. For example, if a folder containing several subfolders with articles, images, etc. is assigned a secondary location, the subfolders with articles, images, etc. will not be automatically available below the new location of the folder.

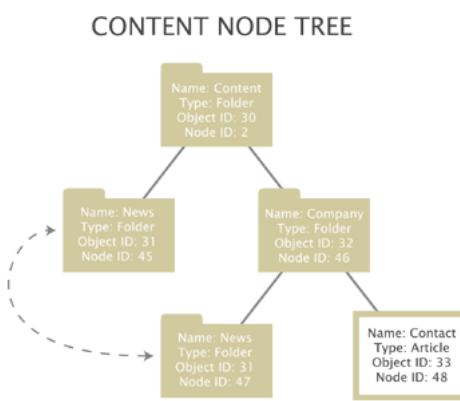


Figure 2.14: Content node tree with multiple locations

Additional notes

Only published objects appear in the tree. A newly created object (status set to draft) does not get a node assignment until the object is published for the first time. An object is considered to be deleted (status set to archived) when all nodes referencing that object are removed from the tree. A deleted object will appear in the system trash. It is important to understand that the trash in eZ publish is a flat structure. This is different from what people are used to from the trash implementation in modern operating systems. When an object is to be recovered/undeleted, it must be manually placed in the tree since the deleted object doesn't contain any information about its previous location. For example, if a folder containing some news articles is deleted, both the folder and the articles will appear on the same level within the trash. Recovering the folder itself will not bring back the articles since the links between the folder and the articles got lost when the nodes were deleted.

2.4.8 Top level nodes

A typical eZ publish installation comes with the following set of top level nodes:

- Content
- Media
- Users
- Setup
- Design

The top level nodes can not be deleted. However, they can be swapped with other nodes. The swap function can be used to change the type of a top level node. For example, the "Content" node references a folder object. By swapping it with another node which refers to a different kind of object, it is possible to change the type of the top level node itself. The following illustration shows the virtual root node and the standard top level nodes:

(see figure 2.15)

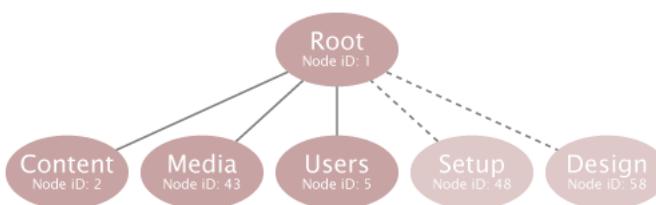


Figure 2.15: Top level nodes

Content

The actual contents of a site is placed under the "Content" node. This node is typically used for organizing folders, articles, information pages, etc. and thus defines the actual content structure of the site. A sitemap can be easily created by traversing the contents of this top level node. The default identification number of the "Content" node is 2. The contents of this node can be viewed by selecting the "Content structure" tab in the administration interface. By default, this node references a "Folder" object.

Media

The "Media" node is typically used for storing and organizing information that is frequently used by the nodes located below the "Content" node. It usually contains images, animations, documents and other files. For example, it can be used to create an image gallery containing

images that are used in different news articles. The default identification number of the "Media" node is 43. The contents of this node can be viewed by selecting the "Media library" tab in the administration interface. By default, this node references a "Folder" object.

Users

The built-in multiuser solution makes use of the native content structure of eZ publish. An actual user is just an instance of a class that contains the "User account" (page 327) datatype. The user nodes are organized within "User group" nodes below the "Users" top level node. In other words, this node contains the actual users and user groups. The default identification number of the "Users" node is 5. The contents of this node can be viewed by selecting the "User accounts" tab in the administration interface. By default, this node references a "User group" object.

Setup

The "Setup" node contains miscellaneous nodes related to configuration and is used internally. The default identification number of the "Setup" node is 48. By default, this node references a "Folder" object.

Design

The "Design" node contains miscellaneous nodes related to design issues and is used internally. The default identification number of the "Design" node is 58. By default, this node references a "Folder" object.

2.4.9 Node visibility

Since publishing means adding an object (by the way of a node) to the content tree, unpublishing would imply the removal of the object from the tree. Once an object is published, it can not be unpublished because eZ publish does not provide such a feature. Instead, the system provides a hiding mechanism which can be used to change the visibility of nodes. The hide feature makes it possible to prevent the system from displaying the contents of published objects. This is achieved by denying access to the nodes. A single node or a subtree of nodes can be hidden either by a user or by the system. A node can have one of the following visibility statuses:

- Visible
- Hidden
- Hidden by superior

All nodes are visible by default and thus the objects they reference can be accessed. A user can hide or unhide a node using the administration interface. Once a node is hidden, all its descendants will automatically be marked "Hidden by superior" and thus the descendants will also become hidden. A node can not become visible if its parent is hidden.

A hidden node will not be available unless the "ShowHiddenNodes" directive within the "[SiteAccessSettings]" block of a configuration override for "site.ini" is set to true. The most common way to use this setting is to disallow all but the administration interface to show hidden nodes.

Implementation

Each node has two flags: "H" and "X". While "H" means "hidden", "X" means "invisible". The hidden flag reveals whether the node has been hidden by a user or not. A raised invisibility flag means that the node is invisible either because it was hidden by a user or by the system. Together, the flags represent the three visibility statuses that were described above:

H	X	Status
-	-	The node is visible.
1	1	The node is invisible. It was hidden by a user.
-	1	The node is invisible. It was hidden by the system because its ancestor is hidden/invisible.

If a user tries to hide an already invisible node then the node's hidden flag will be set in addition to the invisible flag. If a node is hidden and its parent becomes visible, the node will remain hidden while the descendants will remain invisible. The following illustrations show how the node hiding algorithm works.

Case 1: Hiding a visible node

The following illustration shows what happens when a visible node is hidden by a user. The node will be marked hidden. Underlying nodes will be marked invisible (hidden by superior). The visibility status of underlying nodes already marked hidden or invisible will not be changed.

(see figure 2.16)

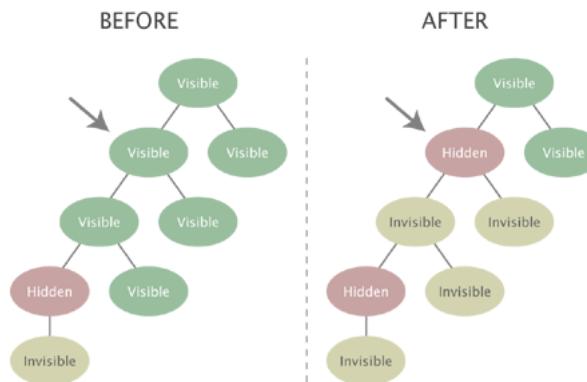


Figure 2.16: Hiding a visible node

Case 2: Hiding an invisible node

The following illustration shows what happens when an invisible node (hidden by superior) is explicitly hidden by a user. The node will be marked as hidden. Since the underlying nodes are already either hidden or invisible, their visibility status will not be changed.

(see figure 2.17)

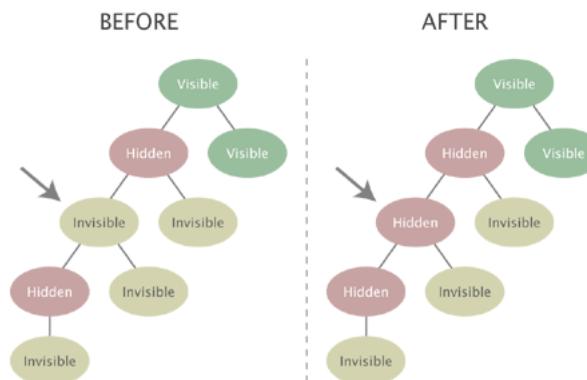


Figure 2.17: Hiding an invisible node

Case 3: Unhiding a node with a visible ancestor

The following illustration shows what happens when a user unhides a node that has a visible ancestor. Underlying invisible nodes will become visible. An underlying node that was explicitly hidden by a user will remain hidden (and its children will be remain invisible).

(see figure 2.18)

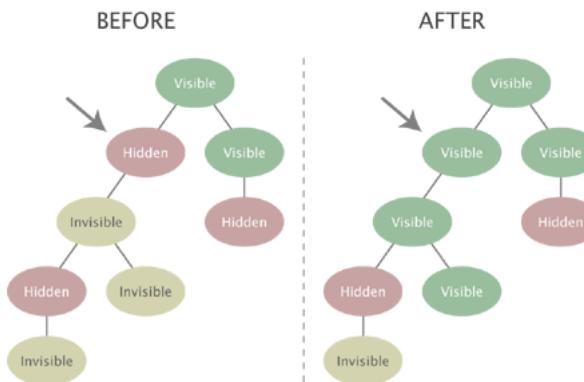


Figure 2.18: *Unhiding a node with a visible ancestor*

Case 4: Unhiding a node with an invisible ancestor

The following illustration shows what happens when a user unhides a node that has an invisible ancestor. Since the target node is unhidet in a subtree that is currently invisible (because a node further up in the hierarchy has been explicitly hidden), the node will not become visible. Instead, it will be marked as invisible and will become visible when the hidden superior node is unhidden.

(see figure 2.19)

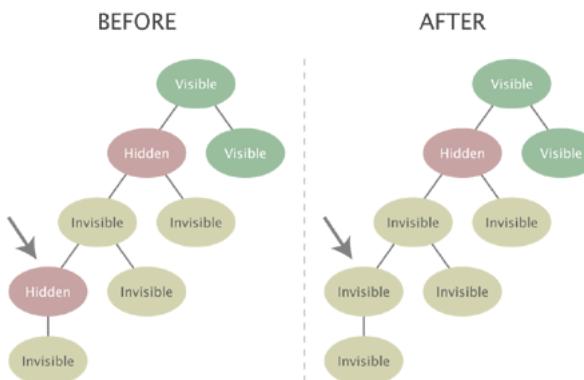


Figure 2.19: *Unhiding a node with an invisible ancestor*

2.4.10 Object relations

The content model of eZ publish makes it possible to create relations between different objects. Any type of object can be connected to any other type of object. This feature is typically useful in situations when there is a need to bind and/or reuse information that is scattered around in the system.

For example, the concept of related objects makes it possible to add images to news articles. Instead of using a fixed set of image attributes, the images are stored as separate objects outside the article. These objects can then be related to the article and used directly in attributes represented by the "XML block" (page 329) datatype. This approach is quite flexible because it does not enforce any limitations when it comes to the amount and the type of information that is to be included.

Relation types

A relation between two objects can be created either at the object level or at the object attribute level. The system stores the different types of relations using the same database table. An object can not have a relation to itself.

Relations at the object level

This method is completely generic and it is always available for use. It allows the users to add objects to an object's related object array. This array is available for all objects; it is nothing more than a collection of the related objects' ID numbers. In other words, the relations can not be grouped in any way.

Relations at the attribute level

This method can be achieved using either the "Object relation" (page 306) or the "Object relations" (page 308) datatypes. While the first one allows only a single relation, the second allows multiple relations. Again, there is no grouping of the relations. However, by making use of several attributes that are represented by one of these datatypes, it is possible to create a structure with grouped relations.

2.4.11 Sections

A *section* is a number that can be assigned to an object. The section ID of an object denotes which section the object belongs to. Each object can belong to one section. By assigning different sections to objects, it is possible to have different groups of objects. Although the sectioning mechanism is implemented at the object level, it is more likely to be used in conjunction with the content node tree. This is why the administration interface makes it possible to manage sections on the node level. Using sections makes it possible to:

- Segment the node tree into different subtrees
- Set up custom template override rules
- Limit and control access to content
- Assign discount rules to a group of products

A default eZ publish installation comes with the following sections:

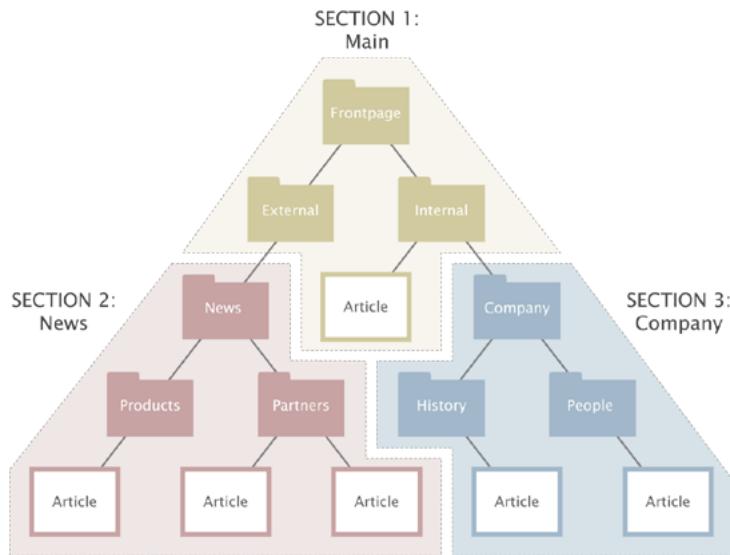
ID	Name	Description
1	Standard	The "Standard" section is the default section. The "Content" top level node makes use of this section.
2	Users	The "Users" section is dedicated for user accounts and user groups that exist on the system. The "Users" top level node makes use of this section.
3	Media	The "Media" section is used by the "Media" top level node.
4	Setup	The "Setup" section is used by the "Setup" top level node.

Section definitions can be added, modified and removed using the administration interface. The following illustration shows an example of how the section feature can be used to segment the content node tree.

(see figure 2.20)

Behavior

When a new object is created, its section ID will be set to the default section (which is usually the standard section). When the object is published, it will automatically inherit the section that is assigned to the object encapsulated by the parent node. For example, if an object is created in

Figure 2.20: *Example of sections.*

a folder that belongs to section 13, the section ID of the newly created object will be set to 13. If an object has multiple node assignments then it is always the section ID of the object referenced by the parent of the main node that will be used. In addition, if the main node of an object with multiple node assignments is changed then the section ID of that object will be updated.

The administration interface makes it possible to assign sections to objects using the node tree. When a section is assigned to a node, the section ID of the object referenced by that node will be updated. In addition, the section assignment of all subsequent children of that node will also be changed. For example, if the section ID of a folder containing news articles is changed, then the section ID of the articles in that folder will also be changed.

The removal of sections may corrupt permission settings, template output and other things in the system. In other words, a section should only be removed if it is completely unused. When a section is removed, it is only the section definition itself that will be removed. Other references to the section will remain and thus the system will most likely be in an inconsistent state. The section ID numbers are not recycled. If a section is removed, the ID number of that section will not be reused when a new section is created.

2.4.12 URL storage

Every address that is input as a link into an attribute using the "XML block" (page 329) or the "URL" (page 325) datatype is stored in a separate part of the database. Actual data stored using these datatypes only contain references to entries in the separate URL table. This feature makes it possible to inspect and edit the published URLs without having to interact with the content objects. The addresses in the URL table can be checked by running the "linkcheck.php" script (which is also executed by the cronjob script) that comes with eZ publish. This script will simply check if the links in the table actually work by accessing them one by one. If the target server of a URL returns an invalid response (404 Page not found, 500 Internal Server Error, 403 Access Denied, etc.) or if there is simply no response, the URL will be marked invalid. Invalid URLs and the objects that are using them can be easily filtered out and edited using the "URL management" part of the administration interface. An entry in the URL table consists of the following data:

- ID
- Address
- Creation time
- Modification time
- Last checked
- Status

Every URL has a unique identification number. The address contains the actual link. The creation time is the exact date/time when the object containing that URL was published. The modification time is updated every time the URL is changed using the URL management part of the administration interface (and not when the object containing that URL is edited). Whenever a URL is checked by the script, the last checked field will be updated. The status of a URL can be either valid or invalid. By default, all URLs are valid. When the cronjob script is running, it will automatically update the status of the URLs. If a broken link is found, its status will be set to "invalid". Whenever an already existing URL is stored, the system will simply reuse the existing entry in the table.

Please note that the link check script must be able to contact the outside world through port 80. In other words, the firewall must be opened for outgoing HTTP traffic from the web server that is running eZ publish.

2.4.13 Information collection

The information collection feature makes it possible to gather user input when a node referencing an information collector object is viewed. It is typically useful when it comes to the creation of feedback forms, polls, etc.

An object can collect information if at least one of the class attributes is marked as an information collector. When the object is viewed, each collector attribute will be displayed using the chosen datatype's data collector template. Instead of just outputting the attributes' contents, the collector templates provide interfaces for data input. The generated input interface depends on the datatype that represents the attribute. The following table reveals the datatypes that are capable of collecting information.

Datatype	Input interface	Input validation
Checkbox (page 272)	Checkbox.	No.
E-Mail (page 278)	Single line of text.	Yes.
Option (page 310)	Radio buttons or a dropdown menu.	No.
Text block (page 319)	Multiple lines of unformatted text.	No.
Text line (page 321)	Single line of unformatted text.	No.

The input interfaces must be encapsulated by an HTML form that posts the data using a submit button named "ActionCollectInformation" to "/content/action" (the "action" (page 496) view of the "content" (page 407) module). The submitted data will be stored in a dedicated part of the database, separated from but related to the object itself. In addition, whenever the object collects any data, the information can be sent to a specified E-mail address. The "Collected information" section within the "Setup" part of the administration interface can be used to view and delete information that was collected through content objects.

2.5 Configuration

This section explains the configuration model of eZ publish. The default configuration files end with a ".ini" extension and are located in the "/settings" directory. Each file controls the behavior of a specific part of the system. For example, the "content.ini" file controls the behavior of the content engine, the "webdav.ini" file controls the behavior of the WebDAV subsystem, and so on. The main and most important configuration file is called "site.ini". Among other things, it tells eZ publish which database, design, etc. that should be used. The default configuration files contain all the possible directives (with default settings) along with brief explanations. These files should only be used for reference. In other words, they should never be modified. The "Configuration files" (page 1132) section of the reference chapter contains a comprehensive explanation of the different configuration files and their settings.

File structure

An eZ publish configuration file is divided into blocks, each block contains a collection of settings. The following example shows a part of the main (site.ini) configuration file.

```
...
# This line contains a comment.
[DatabaseSettings]
Server=localhost
User=allman
Password=qwerty
Socket=disabled
SQLOutput=enabled

# This line contains another comment.
[ExtensionSettings]
ActiveExtensions[]="ezdhtml"
ActiveExtensions[]="ezpaypal"
...
```

The example above shows two blocks: "DatabaseSettings" and "ExtensionSettings". Each block has several settings which control the behavior of the system. A setting can usually be set to enabled/disabled, a string of text or an array of strings. If the name of a setting ends with a pair of square brackets, it means that the setting accepts an array of values. In the example above, the "ActiveExtensions" setting tells eZ publish to use two different extensions: "ezdhtml" and "paypal". Lines starting with a hash are treated as comments.

Configuration overrides

As pointed out earlier, the default configuration files should never be modified because they will most likely be overwritten by a new set of files during an upgrade. Making a backup will still

not be sufficient because the configuration settings change over time. For example, a previous version of the files will not contain settings that were recently added. Because of these issues, custom configuration settings must be placed elsewhere. Global configuration overrides can be placed in the "/settings/override" directory . The settings of the configuration files located in this directory will override the default settings. The name of the configuration files in the override directory must end with one of the following extensions:

- .ini.append
- .ini.append.php

If an override configuration file exist with both ".ini.append" and ".ini.append.php" extensions, eZ publish will process the one which ends with ".php". Because of possible security issues, the latter (.ini.append.php) should be used; specially if eZ publish is running in a non virtual host environment. The ".php" extension will trick the web server into handling the configuration file as a PHP script. If someone attempts to read it using a browser, the server will not display the contents. Instead, it will attempt to process it as PHP, which again will not produce any output since the configuration settings are commented out (see below). This method makes it more difficult for a hacker to get access to the configuration settings (for example the database password) by attempting to access one of the configuration files from outside. In order for this to work, the contents of the configuration file must be encapsulated by a pair of PHP comment markers: /* and */. The following example shows how an override (for example "test.ini.append.php") should be set up:

```
<?php /* #?ini charset="iso-8859-1"?>

# These are my example settings
[ExampleSettings]
ExampleSettingOne=enabled
ExampleSettingTwo=disabled
...
*/ ?>
```

The "charset" directive reveals the character set that was used to construct the ini file (usually ISO-8859-1).

2.5.1 Site management

A single eZ publish installation is capable of hosting multiple sites by making use of something called the *siteaccess system*. This system makes it possible to use different configuration settings based on a set of rules. The rules control which group of settings that should be used in a particular case. The siteaccess rules must be specified in the global override for the site.ini configuration file ("/settings/override/site.ini.append.php").

Siteaccess

A collection of configuration settings is called a *siteaccess*. When a siteaccess is in use, the default configuration settings will be overridden by the settings that are defined for the siteaccess. Among other things, a siteaccess dictates which database, design and var directory that should be used (these are sometime referred to as "resources"). By making use of different siteaccesses, it is possible to combine different content and designs. A typical eZ publish site consists of two siteaccesses: a public interface for visitors and a restricted interface for administrators. Both siteaccesses use the same content (same database and same var directory) but they use different designs. While the administration siteaccess would most likely use the built in administration design, the public siteaccess would use a custom design. The following illustration shows this scenario.

(see figure 2.21)

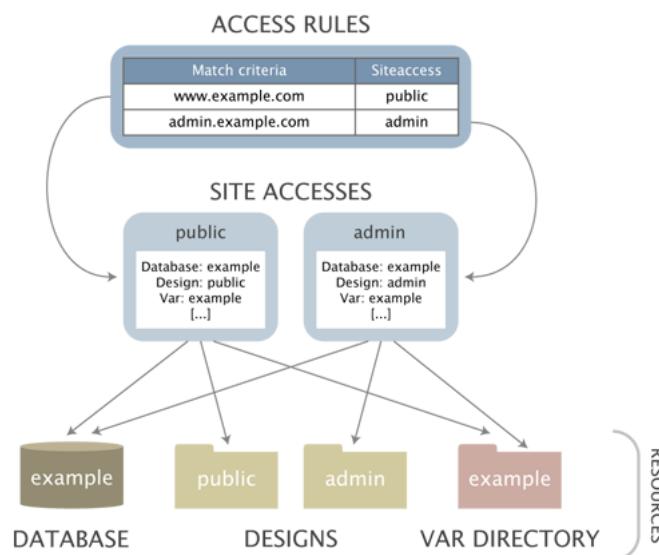


Figure 2.21: Example of a setup with two siteaccesses.

A siteaccess is nothing more than a set of configuration files that override the default settings when the siteaccess is used. A single eZ publish installation can virtually host an unlimited number of sites by the way of siteaccesses. The configuration settings for a siteaccess are located inside a dedicated subdirectory within the "/settings/siteaccess" directory. The name of

the subdirectory is the actual name of the siteaccess. (Please note that siteaccess name should only contain letters, digits and underscores.) The following illustration shows a setup with two siteaccesses: admin and public.

(see figure 2.22)

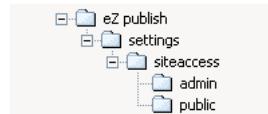


Figure 2.22: Siteaccess directory example.

When a siteaccess is in use, eZ publish reads the configuration files using the following sequence:

1. Default configuration settings (/settings/*.ini)
2. Siteaccess settings (/settings/siteaccess/[name_of_siteaccess]/*.ini.append.php)
3. Global overrides (/settings/override/*.ini.append.php)

In other words, eZ publish will first read the default configuration settings. Secondly, it will determine which siteaccess to use based on the rules that are defined in the global override for "site.ini" ("./settings/override/site.ini.append.php"). When it knows which siteaccess to use, it will go into the directory of that siteaccess and read the configuration files that belong to that siteaccess. The settings of the siteaccess will override the default configuration settings. For example, if the siteaccess uses a database called "Amiga", the system will see this and automatically use the specified database when an incoming request is processed. Finally, eZ publish reads the configuration files in the global override directory. The settings in the global override directory will override all other settings. In other words, if a database called "CD32" is specified in the global override for "site.ini" then eZ publish will attempt to use that database regardless of what is specified in the siteaccess settings. If a setting is not overridden by either the siteaccess or from within a global override then the default setting will be used. The default settings are set by the ini files located in the "/settings" directory. The following figure illustrates how the system reads the configuration files using the "site.ini" file as an example. As already mentioned, settings placed in the override files will be used instead of the default ones.

(see figure 2.23)

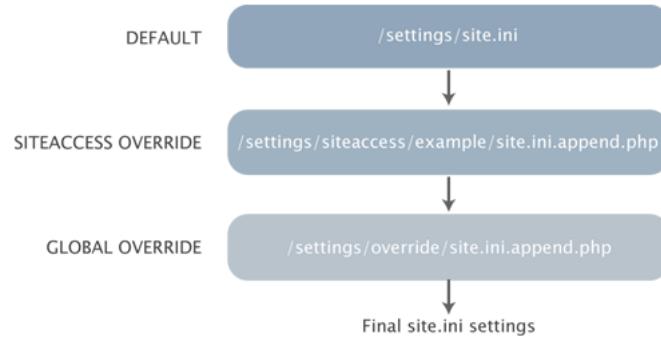


Figure 2.23: Configuration override example.

2.5.2 Extension siteaccess settings

The extension siteaccess settings makes it possible to place siteaccess specific settings in the extensions.

The directory structure must be as follows :

extension/<my_extension>/settings/siteaccess/<my_siteaccess>/<file.ini.append.php>

Example:

extension/ezno/settings/siteaccess/ezno/override.ini.append.php :

```
<?php /*  
  
[article_full_ezno]  
Source=node/view/full.tpl  
MatchFile=article/full.tpl  
Match[class_identifier]=article  
Subdir=templates  
  
*/ ?>
```

Note:

All settings except debug settings and including/activating extensions can be set this way.

2.5.3 Access methods

Based on a set of rules, eZ publish determines which siteaccess it should use every time it processes an incoming request. The rules must be set up in the global override for the site.ini configuration file: "/settings/override/site.ini.append.php". The behavior of the siteaccess system is controlled by the "MatchOrder" setting within the [SiteAccessSettings] block. This setting controls the way eZ publish interprets the incoming requests. There are three possible methods:

- URI
- Host
- Port

The following text gives a brief explanation of the different access methods. Please note that the access methods can be combined. The documentation page of the "MatchOrder" (page 1358) directive reveals how this can be done.

URI

This is the default setting for the "MatchOrder" directive. When the URI access method is used, the name of the target siteaccess will be the first parameter that comes after the "index.php" part of the requested URL. For example, the following URL will tell eZ publish to use the "admin" siteaccess: <http://www.example.com/index.php/admin>. If another siteaccess by the name of "public" exists, then it would be possible to reach it by pointing the browser to the following address: <http://www.example.com/index.php/public>. If the last part of the URL is omitted then the default siteaccess will be used. The default siteaccess is defined by the "DefaultAccess (page 1370)" setting within the [SiteSettings] block. The following example shows how to set up "/settings/override/site.ini.append.php" in order to make eZ publish use the URI access method and to use a siteaccess called "public" by default:

```
...
[SiteSettings]
DefaultAccess=public

[SiteAccessSettings]
MatchOrder=uri
...
```

The URI access method is typically useful for testing / demonstration purposes. In addition it is quite handy because it doesn't require any configuration of the web server and the DNS server.

Host

The host access method makes it possible to map different host/domain combinations to different siteaccesses. This access method requires configuration outside eZ publish. First of all, the DNS

server must be configured to resolve the desired host/domain combinations to the IP address of the web server. Secondly, the web server must be configured to trigger a virtual host configuration (unless eZ publish is located in the main document root). Please refer to the "Virtual Host Setup" (page 70) part of the installation chapter for information about how to set up a virtual host for eZ publish. Once the DNS and the web server is configured properly, eZ publish can be set up to use different siteaccesses based on the host/domain combinations of the incoming requests. The following example shows how to set up "/settings/override/site.ini.append.php" in order to make eZ publish use the host access method. In addition, it reveals the basic usage of the host matching mechanism.

```
...
[SiteAccessSettings]
MatchOrder=host
HostMatchType=map
HostMatchMapItems []=www.example.com;public
HostMatchMapItems []=admin.example.com;admin
...
...
```

The example above tells eZ publish to use the "public" siteaccess if the requested URL starts with "www.example.com". In other words, the configuration files in "/settings/siteaccess/public" will be used. If the requested URL starts with "admin.example.com", then the admin siteaccess will be used. The example above demonstrates only a fragment of the host matching capabilities of eZ publish. Please refer to the reference documentation for a full explanation of the "HostMatchType" (page 1351) directive.

Port

The port access method makes it possible to map different ports to different siteaccesses. This access method requires configuration outside eZ publish. The web server must be configured to listen to the desired ports (by default, a web server typically listens for requests on port 80, which is the standard port for HTTP traffic). In addition, the firewall will most likely have to be opened so that the traffic on port 81 actually reaches the web server. The following example shows how to set up "/settings/override/site.ini.append.php" in order to make eZ publish use the port access method. It also shows how to map different ports to different siteaccesses.

```
...
[SiteAccessSettings]
MatchOrder=port

[PortAccessSettings]
80=public
81=admin
...
...
```

The example above tells eZ publish to use the "public" siteaccess if the requested URL is sent to the web server using port 80. In other words, the configuration files inside "/settings/siteaccess/public" will be used. If the URL is requested on port 81 (usually by appending a :81 to the URL, like this: <http://www.example.com:81>), then the admin siteaccess will be used.

2.6 Modules and views

A *module* offers an HTTP interface which can be used for web based interaction with eZ publish. While some modules offer an interface to kernel functionality, others are more or less independent of the kernel. The system comes with a collection of modules that cover the needs of typical everyday tasks. For example, the content module provides an interface that makes it possible to use a web browser to manage actual content. It is possible to extend the system by creating custom modules for special needs. Custom modules have to be programmed in PHP. The following table gives an overview of some of the most commonly used modules that come with eZ publish.

Module	Description
Content (page 407)	The "Content" module provides an interface to the content engine in the eZ publish kernel. This module makes it possible to display, edit, search and translate the contents of objects, manage the node tree and so on.
User (page 658)	The "User" module provides an interface to the user management system in the kernel. This module makes it possible to log users in and out of the system. In addition, it also provides functionality related to user registration, user activation, password changing, etc.
Role (page 586)	The "Role" module provides an interface to the access control system in the kernel. This module makes it possible to create, modify and delete roles and policies. In addition, it provides functionality for assigning roles to different users and user groups.

Please refer to the "Modules" (page 371) section of the reference chapter for a comprehensive list of all the built-in modules.

Module execution

Every time eZ publish is accessed using a web browser, the client application indirectly interacts with one of the modules that are present in the system. The requested URL tells eZ publish about which module it should execute in order to process the request. In particular, the first part of the URL reveals the name of the module. This is usually the part that comes directly after "index.php" unless the URI access method is used. The following example shows a typical eZ publish URL:

```
http://www.example.com/index.php/content/edit/13/03
```

A quick glance at this URL reveals that the request is directed at the content module. Another typical example of an eZ publish URL could be something like this:

`http://www.example.com/index.php/user/login`

By looking at the URL, we can immediately tell that eZ publish will attempt to execute the user module when processing this request. Obviously, some additional information is also specified in the URLs. In the first example, the name of the module is followed by "/edit/13/03". In the second example, the name of module is followed by "/login". These additional strings control the behavior of the requested module and are explained below.

Module views

A module consists of a set *views*. A view can be thought of as an interface to a module. By using views, it is possible to reach various functions that a module provides. For example, among other things, the content module provides views for displaying, editing, searching and translating the contents of objects. The name of the view that should be accessed appears after the name of the module (separated by a slash) in the URL. In the first example above, eZ publish is instructed to access the "edit" view within the content module. In the second example, eZ publish is instructed to access the "login" view within the user module.

When a view is called, eZ publish starts up the program code that is associated with that view. Upon completion, the view returns a result to the module, which in turn returns it to the rest of the system. The result is put inside a template variable called `$module_result.content` which is available from the main template, the pagelayout. Please refer to the "Template generation" section of the "Templates" chapter for more information about this part of the system.

View parameters

Some views support one or more parameters. A *view parameter* makes it possible to pass information to the view itself and thus allows the view to be controlled from within the requested URL. The view parameters are appended after the name of the view in the URL. In the first example above, the following parameters are passed to the view: "13" and "03". These parameters will instruct the edit view of the content module to provide an interface for editing the third version of the thirteenth content object in the system. The URL given in the second example does not make use of any view parameters. The view mechanism supports two types of parameters:

- Ordered parameters
- Unordered parameters

The ordered parameters have to be separated by slashes and they must come after the name of the view. In addition, they have to be provided in the same order as it is specified in the module's definition. For example, if the view parameters in the first example get mixed up, eZ publish will attempt to edit the thirteenth version of object number three (instead of version number three of object number thirteen).

As the name suggests, the unordered parameters can be provided in an arbitrary order. If the view supports ordered parameters, the unordered parameters must come after the ordered parameters. If the view doesn't support ordered parameters, the unordered parameters will come directly after the name of the view in the URL. The unordered parameters must be provided in pairs. A pair consists of the parameter's name and value separated by a slash. The following example shows an imaginary eZ publish URL with unordered parameters passed to the requested view:

```
http://www.example.com/index.php/video/dvd/button/play
```

The address in the example above tells eZ publish to run the imaginary "video" module and execute the "dvd" view. A variable called "button" will be created and made available for the view code. The value of the variable will be set to "play". It is up to the PHP code of the view to discover this variable and to carry out a necessary sequence of actions.

POST variables

Some views make use of parameters that are submitted by the way of forms through the HTTP POST method. For example, the action view of the content module makes an extensive use of POST variables.

The default request

In order to be able to produce proper output, eZ publish must know which module it should run and which view that should be executed. In other words, every URL has to contain at least the name of an existing module and a view. If an incomplete or mistyped URL is provided, eZ publish will display an error page revealing what's wrong (missing/mistyped module or view). If the requested URL doesn't contain anything after "index.php" (except maybe a slash), the default module/view combination will be executed. The default module/view combination can be configured using the "IndexPage" setting under "[SiteSettings]" in an override for "site.ini". The default setting is "/content/view/full/2". It instructs eZ publish to show a full view of node 2, the content top level node. In other words, if the following request is made:

```
http://www.example.com/index.php
```

...eZ publish will behave as if the following URL was requested:

```
http://www.example.com/index.php/content/view/full/2
```

No redirection or page reload will be made, which means that the address field of the browser will remain unchanged.

2.7 URL translation

This section explains the different URL types that can be used with eZ publish and how the URL translator works. By default, eZ publish is capable of handling two types of URLs:

- System URLs
- Virtual URLs

System URLs

A *system URL* tells eZ publish about which module that should be run and which view that should be executed. It may contain additional parameters/values that are passed to the view itself. Every system URL follows the same structure and can be broken down into the following components:

- Module name
- View name
- View parameters

The view parameters are optional and may consist of ordered and/or unordered values. A comprehensive description of the view parameters can be found in the "Modules and views" (page 149) section. The following model shows the required sequence of the different URL components:

```
http://www.example.com/index.php/<module>/<view>/[<ordered_view_parameters>]/
[<unordered_view_parameters>]
```

URL component	Description
Module	The name of the module that should be run.
View	The name of the view that should be executed.
Ordered view parameters	Some views make use of ordered parameters.
Unordered view parameters	Some views make use of unordered parameters.

The following example shows a typical system URL:

```
http://www.example.com/index.php/content/edit/13/3
```

By looking at the URL, we can tell that it will instruct eZ publish to run the "content" module and execute the "edit" view. The values "13" and "3" are parameters that will be passed to the view itself. Please note that the exact style of the URLs depend on the access method (page 146) that is used and the way the web server is configured. For example, the web server can be configured to hide away the "index.php" part of the address.

Virtual URLs

A *virtual URL* (also known as *URL alias* or *nice URL*) is nothing more than an alias for an existing system URL. Virtual URLs are nicer, easier to remember and sometimes shorter than system URLs. While system URLs reveal a great deal about what eZ publish is instructed to do, virtual URLs do not reveal any system specific information at all. A virtual URL can not be broken down to components in the same way as a system URL. The following example shows a typical virtual URL:

```
http://www.example.com/company/about
```

There are actually two types of virtual URLs, ones that are automatically generated and maintained by eZ publish and ones that are created and maintained by the site administrator. However, all virtual URLs are treated equally and thus they are handled in the same way. The system keeps track of the URLs in a table which basically consists of two columns: virtual/source address and system/destination address. An entry in the URL table might look something like this:

Virtual/source address	System/destination address
company/about	content/view/full/46

An actual URL using the virtual address in the table above could be the following:

```
http://www.example.com/company/about
```

According to the table above, the virtual URL will be translated internally to the following system URL:

```
http://www.example.com/content/view/full/46
```

Both URLs are perfectly valid and will produce the exact same output, in this case a full view of node number 46. When the virtual URL is used, the redirection/mapping will be done internally and thus the user will reach the target node without any glitches in form of redirections, page reloads, etc.

Automated virtual URL generation and maintenance

Every time an object is published, the system will automatically generate a virtual URL for each of the object's node assignments. The generated URL for a node is based on the node's location in the tree and the name of the object that the node encapsulates. The virtual URLs generated for the nodes are handled completely by the system and can not be changed using the administration interface. The following illustration shows an example of objects, nodes and a corresponding URL table.

(see figure 2.24)

The example above clearly demonstrates how the virtual URLs are generated. For each node, the system generates a path of strings separated by slashes. The strings in the path are the names

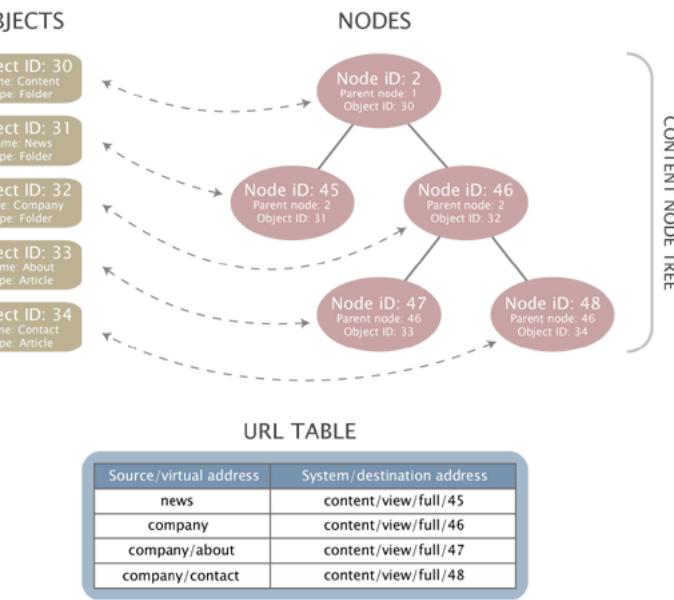


Figure 2.24: *Objects, nodes and the URL table.*

of the objects that are referenced by the nodes up to and including the target node. Special symbols are converted to underscores and special characters are converted using the built in transliteration feature. For example, the Norwegian characters "æ", "ø" and "ÿ" are converted to "ae", "oe" and "aa". If the system is about to generate a virtual URL that already exists, it will simply append an underscore at the end of the newly generated address and thus the risk of having duplicate URLs is eliminated.

When the name of an object is changed, the system will take care of changing the virtual URLs for the involved nodes. In addition, an internal redirection will be created, which will make sure that the old URL still works. The old virtual URL will keep working until the exact same URL needs to be generated for a node. In this case, the old virtual URL will be deleted.

Manual virtual URLs and translations

It is possible to manually add, edit and remove virtual URLs using the administration interface. The URL translator mechanism makes it possible to add three types of translations:

- New virtual URL for an existing system URL
- Secondary / alternative virtual URL for an existing virtual URL
- Wildcard based URL forwarding

URL handling

When eZ publish receives a request, it looks at the URL that was sent by the web browser. The address is stripped for unnecessary parts such as for example the host/domain name, etc. If the address exists as a virtual URL in the table, eZ publish will attempt to process the corresponding system URL. If the address doesn't exist, eZ publish will attempt to interpret it as a system URL.

2.8 Access control

This section explains how eZ publish manages user accounts and access permissions. The system comes with a built-in access control mechanism that can be used to limit access to content or to certain functions. The access control system is based on the following elements:

- User
- User group
- Policy
- Role

The following illustration shows the relations between the elements in the list above.

(see figure 2.25)

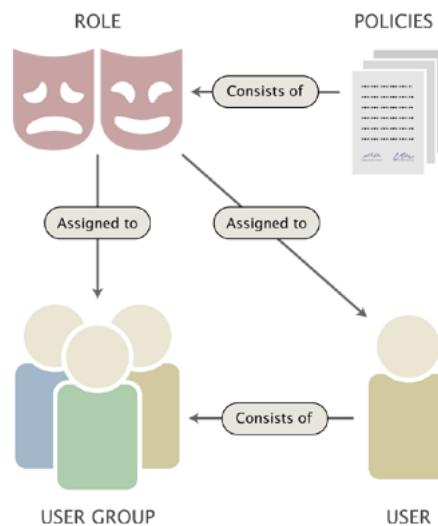


Figure 2.25: *Users, groups, policies and roles.*

A *user* defines a valid user account on the system. A *user group* consists of users and other user groups. A *policy* is a rule that grants access to content or a certain system function. For example, a policy may grant read access to a collection of nodes. A *role* is a named collection of policies. A role can be assigned to users and user groups. The following text gives a more in-depth explanation of the user/group/policy/role elements.

User

An actual user account is represented by a content object (with at least one node assignment) that contains information about a specific user. The default "User" (page 369) class allows the storage

of the following elements: first name, last name, E-mail, username and password. The last three elements (E-mail, username and password) are provided by the "User account" (page 327) datatype. This is a special datatype which plugs more deeply into the system. Instances of any content class containing the "User account" datatype will function as valid users on the system. In other words, if there is a need to store additional information about users, it is possible to either modify the default user class or to create a custom class that contains the datatype. A user account can be enabled or disabled from within the administration interface. When disabled, the account will continue to exist, but the user will not be able to log in until the account is enabled.

Please note that the default configuration does not allow different users to be registered with the exact same E-mail address. This is just a built-in precaution mechanism which can be easily turned off by setting the "RequireUniqueEmail" directive within the [UserSettings] block of a configuration override for "site.ini" to "false".

User ID

Every user has a unique identification number which is the same as the ID number of the actual object that represents the user account. Among other things, the user IDs are used by other objects on the system. In particular, an object contains references (by the way of user IDs) to the initial creator and to all users who have created versions within that object. Removing a user account might lead to an inconsistent state where objects have owner/modifier references to nonexisting user accounts. Because of this, it is not recommended to remove users from the system, the accounts to be removed should be disabled instead.

User group

A user group is a content object (with at least one node assignment) that contains user accounts and other user groups. In other words, a user group is just a collection of users (similar to a directory containing files and subdirectories on a filesystem).

Policy

A policy is a rule that grants access to a specific function or all functions of a module. A policy consists of the following elements:

- Module name
- Function name
- Function limitation

The module name reveals the actual module that the policy grants access to. The function name specifies which function the policy should be limited to. A policy can either be restricted to a single function or grant access to all functions of a module. A module can have none or several functions. The functions are assigned to the module's views and thus the access requirements

for a view are controlled by the functions that are assigned to that view. The function-view assignments can not be tampered with from within the administration interface. A policy granting access to a module's function can be further restricted by the way of function limitations. This can only be done if the function itself supports limitations. A function may support none, one or several limitations. The following table shows an overview of the available function limitations.

Limitation	Description
Class	The "Class" limitation makes it possible to limit a policy to objects of certain types.
Node	The "Node" limitation makes it possible to limit a policy to a specific node.
Owner	The "Owner" limitation makes it possible to limit a policy to objects that are owned by the user who is logged in.
Parent class	The "Parent class" limitation makes it possible to limit a policy based on the type of the object referenced by the parent node.
Section	The "Section" limitation makes it possible to limit a policy to objects that are assigned to certain sections.
Status	The "Status" limitation makes it possible to limit a policy to a certain version status (published, archived, etc.).
Subtree	The "Subtree" limitation makes it possible to limit a policy to a certain part of the content node tree.

Role

A role is a named collection of policies. A role can be assigned to users and user groups. It is possible to assign a role with additional limitations. The role limitation feature is typically useful in a case where multiple users with similar permissions have to manipulate different parts of the content node tree. Instead of creating a role for each user, the site administrator can create a generic role and assign it with different limitations to the different users. The role limitations will override the limitations of the role's policies. The following table shows an overview of the available role limitations.

Limitation	Description
Section	The "Section" limitation makes it possible to limit a role to objects that are assigned to certain sections.
Subtree	The "Subtree" limitation makes it possible to limit a role to a certain part of the content node tree.

2.9 Webshop

This section explains the e-commerce capabilities of eZ publish. The system comes with an integrated shop mechanism that plugs directly into the object / node tree model. The webshop functionality is built around the following components:

- Products
- Value Added Taxes (VATs)
- Discount rules
- Wishlist
- Basket
- Orders

The following illustration shows how the different components interconnect and work together.

(see figure 2.26)



Figure 2.26: The integrated e-commerce solution.

An actual product is represented by a content object (with at least one node assignment) that contains information about the product itself along with a price. The price can be affected by

a value added tax and/or a discount rule. A discount rule can be configured to reduce the price of certain products by a percentage. The products can be put into a user's wishlist and/or shopping basket. A user's wishlist and basket can be modified at any time. The contents of the shopping basket can be purchased by initiating the checkout process. Once the checkout process is completed, an order will be created. The system will automatically notify the site administrator and the user who placed the order by sending out E-mails. A list of placed orders and sales statistics can be viewed using the administration interface. An order is assigned a status which may be changed by a user with sufficient permissions. A status log is kept for each order.

Value added taxes

The system allows the site administrator to set up different kinds of value added taxes (VATs). A VAT consists of a name and a percentage. The administration interface makes it possible to add, remove and modify VATs. The VATs are used by the price datatype.

The price datatype

As pointed out above, a product is nothing more than a content object with a price. The price must be represented by an attribute that makes use of the built-in price datatype. This is a special datatype which plugs more deeply into the system. Instances of any class containing the price datatype will automatically be treated as products. A class attribute represented by the price datatype makes use of one of the predefined VATs. There are two ways in which the selected VAT can be used. This configuration depends on how the product prices are entered when the objects are created. The first alternative (Price inc. VAT) is to be used if the prices that are entered already include the value added tax. The second alternative (Price ex. VAT) should be used if the prices that are entered do not contain the value added tax. When the first alternative is used and the product is viewed, the price that was entered will be shown. When the second alternative is used and the product is viewed, the price will be the price that was entered plus the VAT. When the object is in the basket and the basket is viewed, it is possible to see the price of the products with and without the VATs (regardless of which approach that was used).

Discount rules

The final price of a product can be affected by a discount rule. A discount rule can be configured to reduce the price of certain products by a percentage. The discount rules can be placed in different discount rule groups and are always active (there is no way to turn them on/off). The discount rule groups make it possible to choose which group(s) of customers will be affected. This can be done by assigning a discount rule group to the target user group(s).

By default, a newly created discount rule affects all the products that are in the system. However, a discount rule can be easily limited to a group of products. A discount rule can be limited in two ways, which are mutually exclusive. The first alternative is to use a combination of the "Product type" and the "Section" limitations, which are described in the table below.

Limitation	Description
Product type	The "Product type" limitation makes it possible to limit a policy to products/objects of certain types (only classes that make use of the price datatype will be shown). The default setting is "Any", which means that it will affect all kinds of product objects.
Section	The "Section" limitation makes it possible to limit a policy to products/objects that are assigned to certain sections. The default setting is "Any", which means that the discount rule will affect product objects in all sections.

The second alternative is to add individual products to the discount rule's product list. When the individual product list is used, the "Product type" and "Section" limitations will be omitted and thus only the products that are in the list will be affected.

Shop related datatypes

The following table shows the datatypes that plug in to the e-commerce subsystem of eZ publish.

Datatype	Description
Price (page 312)	When used as an attribute in a content class, the "Price" datatype connects the instances (objects) of that class to the webshop system. As soon as an object has a price attribute, users can put the object in their baskets and/or wishlists. Objects without a price attribute can not be put into a user's basket and/or wishlist and thus they are not connected to the e-commerce subsystem.
Option (page 310)	The "Option" datatype makes it possible to create a single group of options for each content object. Each option can be assigned a short text and an additional price. For example, it can be used to sell T-shirts in different colors where the price is different for some (or all) colors.
Multi-option (page 304)	The "Multi-option" datatype makes it possible to create multiple groups of options for each content object. Each option can be assigned a short text and an additional price. This datatype works in the same way as the "Option" datatype. The only difference is that instead of supporting only one group of options, it allows the creation of multiple groups

	of options for each content object.
Range-option (page 314)	The "Range-option" datatype makes it possible to create a single group of enumerated options for each content object. For example, it can be used in a scenario where the goal is to sell shoes of different sizes and the size does not affect the price. For each content object, the administrator needs to set up the available range (if any).

2.10 Workflows

This section explains the workflow capabilities of eZ publish. The system comes with an integrated workflow mechanism that makes it possible to perform different tasks with or without user interaction. The workflow implementation is based on the following components:

- Events
- Workflows
- Workflow groups
- Triggers

The following illustration shows the relations between the elements in the list above.

(see figure 2.27)

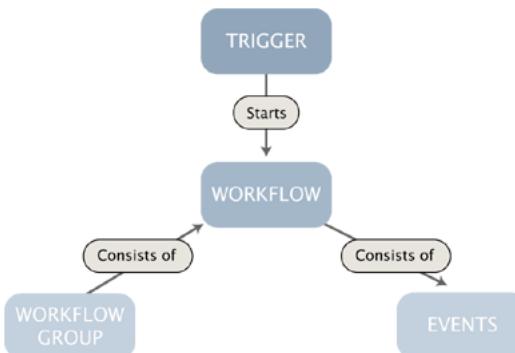


Figure 2.27: The workflow system.

An *event* is the smallest entity of the workflow system, it carries out a specific task. eZ publish comes with a collection of events that cover the needs of typical everyday tasks. For example, the built-in approve event makes it possible to have the contents of an object approved by an editor (a user) before it is published. The built-in events are documented in the "Workflow events" (page 772) section of the "Reference" chapter. It is possible to extend the system by creating custom events for special needs. Custom workflow events have to be programmed in PHP.

A *workflow* is a collection of events. In other words, it defines an ordered sequence of actions that will be executed when the workflow is running. The workflows can be placed in different groups. A *workflow group* is nothing more than a collection of workflows. A workflow is initiated by a *trigger*. Although a trigger is only capable of initiating a single workflow, several other workflows can be started through the built-in multiplexer event (from within the workflow that was originally initiated by the trigger). A trigger is associated with a function of a module. It will start the specified workflow either before or after that the function has completed. The following table gives an overview of the standard/built-in triggers.

ID	Module	Function	Connection type
1	content	publish	before
2	content	publish	after
3	shop	confirmorder	before
4	shop	checkout	before
5	shop	checkout	after

Chapter 3

Templates

The purpose of this chapter is to reveal and teach everything there is to know about the template system. It describes both the template language and the way the system handles the template files. People previously unfamiliar with eZ publish templates should be able to collect enough information in order to understand the following issues:

- What a template is and what it is not
- Template types (pagelayout, node and system templates)
- Template structure
- The template language
- The main template (the pagelayout)
- Template variables available in the pagelayout
- How basic template tasks can be done
- How information can be retrieved from the CMS
- The template override system

3.1 Template basics

This section explains the concepts behind templates and the template system. eZ publish uses templates as the fundamental unit of site design. A *template* is basically a custom HTML file that describes how some particular type of content should be visualized. A template file always ends with a ".tpl" extension. Actual HTML code in the built-in/default templates follow the [XHTML 1.0 Transitional](#) specification. In addition to standard HTML syntax, a template consists of eZ publish specific code. The eZ publish specific code makes it possible to extract information from the system and to solve common programmatic issues like for example conditional branching, looping, etc. All eZ publish specific code must be placed inside a set of curly brackets, "{}". The following example shows a part of a template that prints out the current time:

```
...
<h1>Time machine</h1>
<p>
    The current time is: {currentdate()|l10n( time )}
</p>
...
...
```

The example above demonstrates how standard HTML is mixed with eZ publish specific code. It shows the usage of the "currentdate" (page [806](#)) and the "l10n" (page [821](#)) template operators. Since "currentdate" returns a UNIX timestamp, it must be formatted using the "l10n" localization operator (or else the output would not make any sense to humans). This is done by piping the output from the "currentdate" operator into the "l10n" operator, which will output the requested information according to the current locale settings. The "time" parameter tells the operator to output only the time (it could have been "date", "shortdate", "datetime" and so on).

Template generation

The template system is component based. In other words, an actual HTML page is usually made up of several templates. At the minimum, eZ publish always renders the main template, which is called *pagelayout*. The pagelayout contains the HTML, HEAD and BODY tags; it dictates the overall look of a site. Among other things, it describes the visual structure (main layout, logo, main menu, footer, etc.) that will be presented for each HTML page that the system generates.

Every incoming request tells eZ publish to run a specific module and to execute one of the module's views. When finished, the requested module/view combination will generate a result. The result can be accessed through the \$module_result array which is available in the pagelayout template. The following illustration shows a simplified 3-step explanation of how eZ publish responds to an HTTP request.

(see figure [3.1](#))

Every view generates a chunk of HTML code by making use of a template. Templates that are used by views are often referred to as *view templates*. Whenever a view has finished running, it will issue an internal template request. The requested template will be interpreted, processed and thus converted to HTML. After processing, the system will put the resulting HTML in the

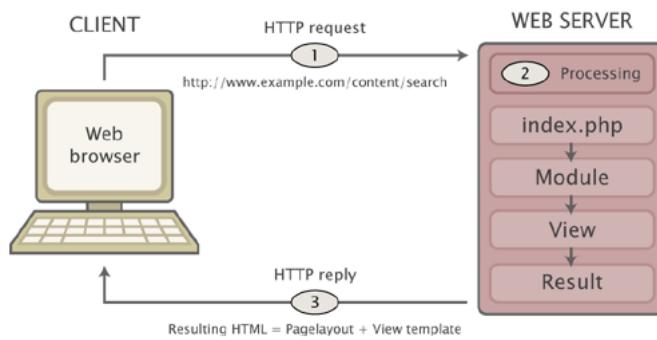


Figure 3.1: Client - server cycle.

module's result array. The module/view's result can be accessed through the ".content" extension: `{$module_result.content}`. By printing out the contents of this variable, it is possible to include the HTML code that was generated by the view in the pagelayout. The following illustration shows how the module/view result (generated by different modules/views - depending on the request) is included in the pagelayout:

(see figure 3.2)



Figure 3.2: The module result as a part of the pagelayout.

View templates

A template used by a view can either be a *node template* or a *system template*. A node template will only be used when a node is being viewed, for example when a system URL containing "/content/view" or the virtual URL of a node is requested. A system template typically provides an HTML interface to a specific eZ publish feature. For example, the template used by the "search" view of the "content" module provides an interface to the built-in search engine.

The difference between the template types mentioned above is the available variables and the combination of override rules that can be used. A node template (page 169) gives access to a vari-

able (`$node`) which contains information about the actual node that is being viewed. Depending on the view that was called, a system template (page 171) typically gives access to several variables. A template override rule makes it possible to display custom templates in specific cases. The override rules for node templates are much more flexible than the override rules for system templates. For example, it is possible to set up complex rule combinations that depend on the type of the node being viewed, the depth of the node in the tree, the section which the node's object is assigned to and so on. Please refer to the "The template override system" (page 222) section for a detailed description of the template override mechanism.

3.1.1 Node templates

Whenever eZ publish is requested to output information about a node (either by a system URL or a virtual URL), it executes the "view" (page 526) view of the "content" (page 407) module. If a system URL is used, both the desired view mode and the target node must be specified in the URL. If a virtual URL is used, eZ publish will automatically know which node that should be accessed by looking up the corresponding system URL in the internal URL table. When a virtual URL is used, the system will always use the full view mode.

The templates for the different view modes must be placed inside the "/templates/node/view/" directory of a design. If the requested file is not found within the main design of the siteaccess, the system will search for it in the additional designs and the standard design. Please refer to the documentation of the automatic fallback system for more information about this feature. The "/templates/node/view" directory of the standard design contains templates for different view modes. A basic custom design typically contains a pagelayout and a full view template. The following illustration shows the locations of these templates in a custom design called "example".

(see figure 3.3)

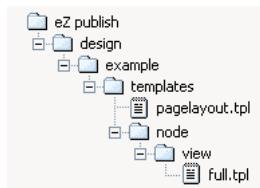


Figure 3.3: Location of pagelayout and full view template in example design.

When a node is requested (and there are no template override rules for node templates), eZ publish will generate a page that is built up of the following templates:

(see figure 3.4)

Custom node templates

A typical eZ publish site always makes use of custom node templates. The main reason for this is because there is almost always a need for displaying the various types of nodes in different ways. For example, information pages need to look different than news articles; the welcome page has to be formatted in a special way, and so on. Unlike custom system templates (which are mostly just modified copies of the standard templates placed in a custom design), custom node templates are created as override templates. The override templates are triggered by the template override system. This system offers a flexible mechanism that can be programmed to use different templates based on various conditions. For example, it can be programmed to use a template called "article.tpl" when the system is requested to show the contents of nodes referencing article objects and at the same time show "special_article.tpl" when a specific article is accessed. Please refer to the documentation of the template override system (page 222) for more information about how this mechanism actually works and how it can be used to trigger



Figure 3.4: Pagelayout + node view full template.

override templates.

The \$node variable

Whenever the system makes use of a node template (regardless of the view mode, the target node and if the template is an override or not), a variable called \$node will be available in the template that is used. This variable is automatically set by the system and it contains an `ezcontentobjecttreenode` (page 721) object that represents the requested node. This variable allows the extraction and display of various information about the node and the object that it encapsulates. Please refer to "Outputting node and object data" (page 219) for information about how to display node/object data.

3.1.2 System templates

Whenever eZ publish is requested to do something else than displaying a node (in other words the URL does not contain ”/content/view” or isn’t the virtual URL of a node), it will use a system template. There are two main differences between system templates and node templates:

- System templates provide access to various variables (depending on the view that was requested). A node template only provides access to a \$node variable representing the node that was requested.
- The override rules for node templates are much more flexible than the override rules for system templates.

An eZ publish distribution provides default templates for all views. These templates are located in the ”templates” directory of the standard design. A view typically uses a template that is located in a subdirectory that has the same name as the module which the view belongs to. The name of the template is usually the same as the name of the view (with a ”.tpl” extension). For example, the ”login” view of the ”user” module looks for a template called ”login.tpl” inside a directory called ”user”. Another example would be the ”basket” view of the ”shop” module. This view looks for a template called ”basket.tpl” within the ”shop” directory.

Custom system templates

Although eZ publish provides all the necessary system templates (by the way of the standard design), a typical eZ publish site always makes use of customized system templates. The main reason for this is because the default templates usually need to be tailored in order to fit perfectly in with the style of a custom design. Unlike custom node templates which are mostly provided using the template override system, custom system templates are usually just modified copies of the standard templates located in the custom design. For example, a custom template for the ”login” view of the ”user” module in a design called ”example” would be ”/design/example/templates/user/login.tpl”. A custom template for the ”search” view of the ”content” module would be ”/design/example/templates/content/search.tpl”.

Design combinations

As mentioned in the text above, a custom design typically contains a set of customized system templates. However, creating a custom design that provides templates for all possible scenarios would be too much / unnecessary work. This is why the standard design always should be used as the last fallback resort. The automatic fallback system makes it possible to combine several designs so that the main design (which is usually a custom design) does not have to provide all the necessary templates. Whenever eZ publish is unable to find a template within the main design of the siteaccess, the system will look for it in the additional designs and the standard design.

Commonly used system templates

The following table shows some of the most commonly used system templates.

Request	URL	Module	View	Template
Search interface	/content/search	content	search	/templates/content/search.tpl
Shopping basket	/shop/basket	shop	basket	/templates/shop/basket.tpl
Login page	/user/login	user	login	/templates/user/login.tpl
User registration	/user/register	user	register	/templates/user/register.tpl

3.2 The pagelayout

The pagelayout is the main template. Among other things, it dictates the overall look of a site. The filename of the pagelayout template must be "pagelayout.tpl". It has to be placed inside the "templates" directory of a design. If eZ Publish is unable to find a pagelayout within the current design (specified by the siteaccess), it will attempt to use the pagelayout template that is provided by one of the fallback designs. The following illustration shows the location of the pagelayout template located in a design called "example".

(see figure 3.5)



Figure 3.5: The location of the pagelayout (main) template.

The pagelayout contains the HTML, HEAD and BODY tags (the other HTML framework). In addition, it dictates the overall look of a site. Among other things, it is used to describe the visual structure (main layout, logo, main menu, footer, etc.) that will be presented for every page request. The following example shows what is considered to be the most basic pagelayout:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

<style type="text/css">
    @import url({'stylesheets/core.css'|ezdesign});
    @import url({'stylesheets/debug.css'|ezdesign});
</style>

{include uri='design:page_head.tpl'}

</head>

<body>

{$module_result.content}

</body>
</html>
```

The document type

The very first line in the pagelayout is used to declare the document type of the pages that are generated by eZ Publish. Per HTML and XHTML standards, a DOCTYPE (short for "document type declaration") informs browsers and syntax validation engines about which version of (X)HTML that is used. This information should be included at the very top of in every web page, this is why it is the first part of the pagelayout.

The DOCTYPE declaration is one of the key components when it comes to proper rendering and compliant web pages. A DOCTYPE that includes a full URL tells the browser to render the page in standards-compliant mode, treating the (X)HTML, CSS, and DOM structures as they should be treated according to the standards. A missing, incomplete or outdated DOCTYPE throws most browsers into something called "Quirks" mode. In this mode, the browser assumes that the document was written using old-fashioned, invalid markup and code per the chaotic industry norms of the late 1990s. In other words, the page will most likely not be rendered according to the standards and it will certainly not validate.

The HTML tag

The HTML tags encapsulate the marked up contents of an actual web page. In addition to the tag itself, the HTML tag in the example above includes a URL to the XHTML specification. XHTML is a family of current and future document types and modules that reproduce, subset, and extend HTML 4. The XHTML family document types are XML based, which means that they are designed to work in conjunction with XML-based user agents.

In document processing, it is often useful to identify the natural or formal language in which the content is written. The "lang" and "xml:lang" attributes specify the language of the entire HTML element. The value of the xml:lang attribute takes precedence. The language values should be set to the language that is used throughout the site. The values of the attributes are language identifiers as defined by ISO 3166-1 (and the corresponding ISO 3166-1-alpha-2) standards.

The head tag

The head tag contains information about the document itself. The information contained here doesn't show up on the page displayed in a web browser. Only the contents of the title tag will be made visible (as the title of the browser window). The head tag typically contains information about which CSS files that should be used, a description of the document itself, keywords and so on.

Cascading Style Sheets

The pagelayout in the example above makes use of two CSS files: "core.css" and "debug.css". The code encapsulated by curly brackets is eZ Publish specific code. What happens here is that the text within the quotes is piped into a template operator called "ezdesign" (page 962). The operator prepends the text with the path to the current design directory (the one which is specified using

the "SiteDesign" configuration directive). This technique assures that the path to the CSS files are always correct, regardless of the access method (page 146) that is being used. For example, if the name of the current design is "my_design" and it includes a CSS file called "example.css", the following output will be produced:

```
@import url("/design/my_design/stylesheets/example.css");
```

The "core.css" and "debug.css" files are a part of the standard design that comes with eZ publish. It is not necessary to have these CSS files in the "stylesheets" directory of a custom design. If eZ Publish is unable to find the files within the current/custom design, it will automatically use the ones that are in the standard design. Please refer to the description of the automatic fallback system for a detailed description of the fallback mechanism. Because of the fallback system, the style-part of the pagelayout presented above will most likely result in the following output:

```
...
<style type="text/css">
    @import url("/design/standard/stylesheets/core.css");
    @import url("/design/standard/stylesheets/debug.css");
</style>
...
```

The core stylesheet

The "core.css" file defines a standard set of basic styles (font styles, sizes, margins, etc.) for both general HTML elements and some eZ Publish specific classes. The eZ Publish specific classes are used by the standard templates. A site that makes an extensive use of the default templates should always have the "core.css" file included in the pagelayout. Otherwise, the missing styles may cause the unexpected rendering of various elements.

The standard "core.css" file should never be changed. If there are basic styles in core.css that doesn't fit the visual environment of a site, a modified version of "core.css" may be placed in the custom design that the site uses. However, the recommended solution is to create a completely new CSS file that contains both custom classes and overrides for elements defined in "core.css".

The debug stylesheet

The "debug.css" file contains styles that are used to format the debug output which appears at the bottom of the page when debug output is enabled. The usage of the "debug.css" file is only necessary during the development of the site (typically when debug information is needed) and thus it can be removed or commented out before the site is launched.

Document information

The system is capable of automatically generating information about the page itself (title, meta tags, keywords, etc.). This can be done by the inclusion of the page head (page 177) template

("page_head.tpl"), which is located in the templates directory of the standard design. If eZ Publish is unable to find the requested file in current/custom design, it will automatically fallback and use the file located in the standard design.

The body tag

The body tag defines the document's body, which contains the actual contents of the web page (text, images, etc.) marked up in an orderly fashion. At the minimum, an eZ Publish pagelayout should contain the result from the requested modules.

Module result

Upon every request, eZ Publish automatically generates an array called "module_result". This array is available only in the pagelayout template. It contains all the necessary information about which module that was run, which view that was called, the output that was produced and so on. The actual output (for example the contents of a news article) can be included in the pagelayout by accessing the "content" element of the \$module_result array, the syntax is:

```
{$module_result.content}
```

When the pagelayout is rendered, the {\$module_result.content} part will be replaced with the actual output that the requested module produced. Please refer to the "Variables in pagelayout" (page 181) page for an overview of the template variables that can be accessed from within the pagelayout.

Debug information

The last part of a typical eZ Publish pagelayout is an HTML comment that looks like this:

```
<!--DEBUG_REPORT-->
```

If the debug information is turned on, eZ Publish will replace this comment with the actual debug report when the pagelayout is processed. In other words, the debug report will be included as a part of the generated page and thus it will not cause invalid output by breaking the HTML structure. The debug reports that eZ Publish generates follow the XHTML 1.0 Transitional specification and thus the debug information validates.

3.2.1 The page head

The standard design contains a page head template that can be used to automatically generate important tags that should be included in the head section of every HTML response. The output of the standard head template (/design/standard/template/page_head.tpl) can be broken down into the following group of tags:

- Title tag
- Meta tags
- Link tags

The following HTML dump shows an example of the output from the standard page head template.

```
<title>Current / Parent / Top - Site name</title>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="Content-language" content="eng-GB" />
<meta name="author" content="eZ Systems" />
<meta name="copyright" content="eZ Systems" />
<meta name="description" content="Content Management System" />
<meta name="keywords" content="cms, publish, e-commerce, content management,
development framework" /
>
<meta name="MSSmartTagsPreventParsing" content="TRUE" />
<meta name="generator" content="eZ publish" />

<link rel="Home" href="/" title="Front page" />
<link rel="Index" href="/" />
<link rel="Top" href="/" title="Current / Parent / Top - Site name" />
<link rel="Search" href="/content/advancedsearch" title="Search Site name" />
<link rel="Shortcut icon" href="/design/standard/images/
favicon.ico" type="image/x-icon" />
<link rel="Copyright" href="/ezinfo/copyright" />
<link rel="Author" href="/ezinfo/about" />
<link rel="Alternate" href="/layout/set/print/content/view/full/
64" media="print" title="Printable version" />
```

Title

The contents of the title tag is based on the location being viewed (the location within either the content node tree or the system itself) and the actual name of the site. The path to the element being viewed is reversed and thus the current element becomes the first component of the title. The components of the path are separated by slashes. When a node is viewed, the path elements

will be the actual names of the objects which are encapsulated by the nodes that make up the path up to and including the target node. When a system function is being accessed (for example the login view of the user module: "/user/login"), the path will most likely be a reversed version of the module/view combination that was used. The name of the site is appended at the end of the path, separated by a dash. The site name can be configured using the "SiteName" directive in a configuration override for "site.ini".

The example above demonstrates the output of the page head template when a node is being viewed. The name of the object encapsulating the node is "Current". The name of the other objects (encapsulated by the parent node and so on) are "Parent" and "Top". The name of the site is "Site name".

Meta tags

In addition to the actual information contained on a web page, the HTML of the page may also include information about the document itself. This is achieved by making use of so called meta tags. The information given by meta tags is usually not visible when the web page is viewed. However, the meta tags are used by the web browser and miscellaneous search engines that index and rank the contents of web pages. The standard page head template outputs the most commonly used meta tags. It can be broken down into three types of tags:

- HTTP-EQUIV meta tags
- Generic meta tags
- Additional meta tags

HTTP-EQUIV meta tags

Meta tags with an HTTP-EQUIV attribute are equivalent to HTTP headers. These tags usually control the way a browser interprets the document. Tags using this form should have an equivalent effect when specified as an HTTP header. Some web servers automatically translate the contents of these tags to actual HTTP headers. The HTTP-EQUIV meta tags in the page head make sure that the browser (and also search engines) know which character set and language the document uses. The language and character set values are automatically set by eZ publish based on the language and character set that the site uses.

Generic meta tags

The generic meta tags make it possible to reveal meta information about the document itself. Although the specification of meta tags does not define a set of legal meta data properties, it is a common practice to include generic information such as the name of the author, description of the site, copyright notices, keywords, etc. By making use of the "MetaDataArray[...]" directive in a configuration override for "site.ini", the site administrator can set up a custom set of generic meta tags. eZ publish will loop through and display the name and value of the specified tags. The

example above shows the default meta tags that will be used if no custom meta tag configuration is present.

Additional meta tags

The last meta tags set by the standard page head template prevent the usage of smart tags and reveal the name of the software that was used to generate the output.

Link tags

Link tags in the HTML head make it possible to relate the document to other documents. This is done by the way of REL and REV attributes. While REL links are used to establish relationships, REV links are used to establish reverse relationships. Some browsers make use of the link tags in order to produce a navigation bar that can be used to quickly navigate the site. The links tags generated by eZ publish are specified in the "link.tpl" file within the templates directory of the standard design. The standard page head makes use of the "links.tpl" file. The default output of the standard page head template produces a basic set of links that can be used to navigate to different parts of the site. The following list shows the link tags that the page head generates:

Link	Description
Home	The "Home" link points to the root/start of the site. It will always bring the user back to the front page (for example http://www.example.com).
Index	The "Index" link points to the root/start of the site. It will always bring the user back to the front page (for example http://www.example.com).
Top	The "Top" link points to the root/start of the site. It will always bring the user back to the front page (for example http://www.example.com).
Search	The "Search" link points to the "advanced search" view of the "content" module. It will bring the user to the advanced search interface (http://www.example.com/content/advancedsearch).
Shortcut icon	The "Shortcut icon" defines the location of the favorite/shortcut icon. Most browsers will display this icon in front of URLs in the address field and in the bookmark list. The default shortcut icon is the double square white-orange eZ Systems logo. It can be easily replaced by putting a 16x16 pixel icon file (16 color BMP/Windows Icon Format) called "fav-

	icon.ico" in the images folder of a site design.
Copyright	The "Copyright" link points to the "copyright" view of the "ezinfo" module. The default copyright page of eZ publish will be displayed (http://www.example.com/ezinfo/copyright.).
Author	The "Author" link points to the "about" view of the "ezinfo" module. The default about page of eZ publish will be displayed (http://www.example.com/ezinfo/about.).
Alternate	The "Alternate" link points to a alternate/printerfriendly version of the page. The printerfriendly version of a page is achieved by making use of the "set" view of the "layout" module. This technique makes it possible to use an alternative pagelayout which is usually stripped for everything (menus, logos, etc.) except the actual content that is being presented.

Link parameters

The links can be completely turned off by passing "enable_link=false()" when including the page head template:

```
{include uri='design:page_head.tpl' enable_link=false()}
```

The link to the alternate/print layout can be turned off by passing "enable_print=false()" when including the page head template:

```
{include uri='design:page_head.tpl' enable_print=false()}
```

3.2.2 Variables in pagelayout

The pagelayout template contains miscellaneous variables that can be used to display information about the state of the system and/or to control the output. The following table shows the available variables along with a brief description.

Variable	Type	Description
\$access_type	array	The name of the siteaccess (as "name") and the ID number (as "type") of the access method (page 146) that was used (1=URL, 2=Host, 3=Port).
\$anonymous_user_id	integer	The ID number of the content object that represents the anonymous user account (the default/standard value is 10).
\$current_user	object	The ezuser (page 765) object of the user who is currently logged in. If no user is logged in, the anonymous user account will be used.
\$ezinfo	array	An array of three strings: "version", "version_alias" and "revision". These strings reveal basic information about the eZ publish release that is being used.
\$module_result	array	Contains information about the result (and the result itself) generated by the module/view that was executed.
\$navigation_part	array	A hash containing the name and the identifier (the keys are "name" and "identifier") of the current navigation part; for example: "Content structure" and "ezcontent-navigationpart". The navigation part is used by the administration interface to determine which part the user interacts with.
\$requested_uri_string	string	Contains the site specific part of the requested URL, for ex-

		ample: "content/view/full/44" (system URL) or "company/about" (virtual URL).
\$site	array	Contains miscellaneous information about the siteaccess that is being used (site name, design resource, meta tags, etc.)
\$ui_component	string	The user interface component which eZ publish uses while the current page is being shown. This variable is used by the administration interface.
\$ui_context	string	The user interface context in which eZ publish is in while the current page is being shown. This variable is used by the administration interface to distinguish between different modes (for example "navigation", "edit", "browse", etc.).
\$uri_string	string	The system version of the requested URL (for example "/content/view/full/13").
\$warning_list	array	An array of warnings related to problems that were discovered when the page was rendered.

\$module_result

The \$module_result array contains the result that was generated by the module/view which was executed. If eZ publish was instructed to display the contents of a node, the variable will contain additional information about the node that was requested. If eZ publish was instructed to do something else (practically anything that is not an actual node view), the result will not contain additional information. The following tables show the contents of the \$module_result variable in the different scenarios.

The default \$module_result

Element	Type	Description
content	string	The actual content (result of

		templates) that was generated by the requested view.
path	array	<p>An array of hashes containing information about the path which leads to the page that is currently being viewed. Each hash contains the following keys: "text", "url". The "text" element usually contains the name of the module/view (for example "Collected information"). The "url" element contains the address. The "url" key of the last element in the array is usually set to false.</p> <p>The standard page head (page 177) template uses the path array to build the TITLE component of the HEAD section. In addition, the path array can for example be used to build breadcrumbs (a path with names (as hyperlinks) of pages/views that lead to the current page/view).</p>
is_default_navigation_part	boolean	Returns TRUE if the default navigation part is being used (the one which is set in PHP code). Returns FALSE if the navigation part of the current module/view has been reconfigured by the site administrator. This can be done by making use of the "NavigationPart" directive of the "[ModuleSettings]" section within a configuration override for "module.ini".
navigation_part	string	The identifier of the current navigation part (for example "ezcontentnavigation-part"). This variable is used by the administration interface to determine which part

		the user interacts with.
ui_context	string	The user interface context in which eZ publish is in while the current page is being shown. This variable is used by the administration interface to distinguish between different modes (navigation, edit, browse, etc.)
ui_component	string	The user interface component which eZ publish uses while the current page is being shown. This variable is used by the administration interface.
uri	string	Contains the site specific part of the requested URL, for example: "content/view/full/44" (system URL) or "company/about" (virtual URL).

The \$module_result when a node is being viewed

Element	Type	Description
content	string	The actual content (result of templates) that was generated by the requested view.
view_parameters	array	An array of the parameters that were sent to the view (for example "limit", "offset", etc.).
path	array	An array of hashes containing information about the path of nodes which lead to the node that is currently being viewed. Each hash contains the following components: Key: text Description: The name of the object referenced by the node. Key: url Description: The system

		<p>URL of the node (for example ”/content/view/full/44”).</p> <p>Key: url_alias Description: The virtual URL of the node (for example ”company/about_us”).</p> <p>Key: node_id Description: The ID number of the node.</p> <p>The node being viewed will have its ”url” and ”url_alias” components set to false. In addition, the ”node_id” will not be available. The path array can for example be used to build breadcrumbs (a path with names (as hyperlinks) of the objects referenced by the nodes that lead to the target/current node).</p>
title_path	array	Almost the same as the ”path” array (see above). When a node is being viewed, the standard page head (page 177) template uses the ”title_path” array to build the TITLE component of the HEAD section.
section_id	string	The ID number of the section which the object referenced by the node being viewed belongs to.
node_id	string	The ID number of the node that is being viewed.
navigation_part	string	Contains the name identifier of the current navigation part (for example ”ezcontentnavigationpart”). This variable is used by the administration interface to determine which part the user interacts with.
content_info	array	Contains miscellaneous information about the node

that is being viewed:

Variable: node_id

Type: string

Description: The ID number of the node.

Variable: parent_node_id

Type: string

Description: The ID number of the parent node.

Variable: object_id

Type: string

Description: The ID number of the object referenced by the node.

Variable: class_id

Type: string

Description: The ID number of the class which the object is an instance of.

Variable: class_identifier

Type: string

Description: The identifier of the class which the object is an instance of (for example "forum_message").

Variable: offset

Type: integer

Description: The offset view parameter.

Variable: viewmode

Type: string

Description: The view mode that was used to display the node (for example "full", "line", etc.).

Variable: node_depth

Type: string

Description: The depth of the node in the content tree.

		<p>Variable: url_alias Type: string Description: The virtual URL of the node (for example "company/about_us").</p> <p>Variable: persistent_variable Type: n/a Description: A variable set in one of the templates used by the view that was executed. Regardless of the caching mechanisms used, this variable will be available in the pagelayout. The type of the persistent variable depends on the value it contains. If the variable is not set, it will simply return a boolean FALSE.</p> <p>Variable: class_group Type: array Description: The ID numbers of the class groups that the class (which the object being viewed is an instance of) belongs to. This variable is connected with a feature that makes it possible to create template overrides based on class groups. By default the "class_group" always returns a boolean FALSE value because the class group override feature is turned off. It can be turned on by setting the "EnableClassGroupOverride" directive in the [ContentOverrideSettings] block of a configuration override for "content.ini" to "true".</p>
cache_ttl	integer	The TTL (Time To Live) value

		of the result that was generated by the module's view (as seconds). A TTL of minus one means that the view cache should never expire. A TTL of zero means that the result should never be cached.
is_default_navigation_part	boolean	Returns TRUE if the default navigation part is being used (the one which is set in PHP code). Returns FALSE if the navigation part of the current module/view has been reconfigured by the site administrator. This can be done by making use of the "NavigationPart" directive of the "[ModuleSettings]" section within a configuration override for "module.ini".
ui_context	string	The user interface context in which eZ publish is in while the current page is being shown. This variable is used by the administration interface to distinguish between different modes (navigation, edit, browse, etc.)
ui_component	string	The user interface component used by eZ publish while the current page is being shown. This variable is used by the administration interface.
uri	string	The site specific part of the requested URL, for example: "content/view/full/44" (system URL) or "company/about" (virtual URL).

3.3 The template language

The eZ publish template language makes it possible to extract information from the system and to solve common programmatic issues like for example conditional branching, looping, etc. All eZ publish specific code must be placed inside a set of curly brackets, "{}". A template file is a combination of HTML and eZ publish template code. Everything that is encapsulated by curly brackets will be interpreted by the template parser when the template is processed. Everything outside the curly brackets will be ignored and thus it will be sent to the browser without any changes.

Curly bracket issues

Since curly brackets are reserved for defining blocks of eZ publish template code, these characters can not be used directly in a template. For example, Javascript code can not be inserted directly into a template file because it makes an extensive use of curly brackets. All non template specific code/text that uses curly brackets must be put inside a "literal" section. The contents of a literal section will be ignored by the template parser. The following example demonstrates the usage of the literal tags:

```
...
{literal}
<script language="JavaScript" type="text/javascript">
<!--
    window.onload=function()
    {
        document.getElementById( 'sectionName' ).select();
        document.getElementById( 'sectionName' ).focus();
    }
-->
</script>
{/literal}
...
```

Outputting curly brackets

It is possible to output curly brackets using two template functions called "ldelim" (page 1013) and "rdelim" (page 1015) (short for left delimiter and right delimiter). The following example demonstrates the usage of these functions:

```
...
This is the left curly bracket: {ldelim} <br />
This is the right curly bracket: {rdelim} <br />
...
```

The following output will be produced:

This is the left curly bracket: {
This is the right curly bracket: }

3.3.1 Comments

Just like in almost any programming language, comments can be used to add explanations, descriptions, etc. Template comments are ignored by the parser and will not be displayed in the resulting HTML output.

There is only one way to add template comments, and that is by encapsulating a block of code by a matching pair of the "{*" and "*}" sequence of characters (left curly bracket + asterisk and asterisk + right curly bracket). In other words, a template comment is just like any other template code except that the curly brackets are accompanied by adjacent asterisks. It is possible to comment both single and multiple lines of code. However, nesting of comments is not supported (it is not possible to comment a chunk of code that already is a comment). The following examples demonstrate the use of comments.

Single line comment

```
{* This is a single line comment. *}
```

The example above will not produce any output.

Multi-line comment

```
{* This is a long comment that  
spans across several lines  
within the template file. *}
```

The example above will not produce any output.

Nested comments (illegal)

```
{* {* Nested comments are not supported! *} *}  
This text will be displayed. *}
```

The example above will produce the following output:

```
This text will be displayed.
```

3.3.2 Variable types

The eZ publish template language supports the following variable types:

- Numbers
- Strings
- Booleans
- Arrays
- Objects

While some variable types can be created on the fly, others need to be created using an operator. Types that may be created directly are numbers and strings. Booleans and arrays must be created using operators, objects may be created using miscellaneous functions and operators. In addition to the types listed above, it is also possible to create and use custom variables. Custom variable types must be represented as objects.

Numbers

Numbers are numerical values. A number can be a positive or a negative integer or a floating point value. The following example demonstrates how different numbers can be used directly within template code:

```
{13}  
{1986}  
{3.1415}  
{102.5}  
{-1024}  
{-273.16}
```

Strings

A string is an arbitrary sequence of characters (text) that is encapsulated by a matching pair of either single or double quotes, ' or ". If the quotes are omitted, the string will most likely be interpreted as a function name. Strings are usually defined in the following way:

```
{'This is a string.'}  
{"This is another string."}
```

The output of the example above would be:

This is a string.
This is another a string.

Using quotes

It is possible to use quotes inside strings. This can be done by either using a different kind of quote or by making use of the escape character (backslash). The following examples demonstrate the use of quotes inside strings:

```
{'The following text is double quoted: "Rock and roll!"  '}  
{"The following text is single quoted: 'Rock and roll!'  "}  
{'Using both single and double quotes: "Rock'n roll!"  '}  
{'Using both single and double quotes: 'Rock'n roll!\'  '}  
{"Using both single and double quotes: 'Rock'n roll!\'"  "}  
{"Using both single and double quotes: "Rock'n roll"  "}
```

The output of the example above will be:

```
The following text is double quoted: "Rock and roll!"  
The following text is single quoted: 'Rock and roll!'  
Using both single and double quotes: "Rock'n roll!"  
Using both single and double quotes: 'Rock'n roll!'  
Using both single and double quotes: 'Rock'n roll!'  
Using both single and double quotes: "Rock'n roll!"
```

Because of the way template code is defined (encapsulated in a matching pair of curly brackets), the right curly bracket, "}", must also be prepended by the backslash escape character. The following example demonstrates this.

```
{'{ This text is inside curly brackets.}'}
```

The output of the template code above will be:

```
{This text is inside curly brackets.}
```

Template strings do not support inline expansion of variables (as in Perl and PHP). In other words, it is not possible to mix variables into strings. However, the concat (page 924) operator can be used to append the contents of some variable to a string; which means that this operator can be used to build strings consisting of other strings and/or miscellaneous variables.

Booleans

Booleans are binary, they are either TRUE (1) or FALSE (0). A boolean must be created using either the "true" (page 864) or the "false" (page 846) template operator. Example:

```
{true()
{false()}
```

For some operators and functions, it is possible to use integers as booleans. However, these are not "real" booleans. Zero means FALSE; all non-zero values mean TRUE. Some operators are able to treat an array as if it were a boolean value. While an empty array means FALSE, a non-empty array means TRUE.

Arrays

Arrays are containers that are capable of holding a collection of any other variable type including other arrays. An array can be a simple vector or a hash map (associative array). An element of a vector can be accessed using an index number. The number denotes the position of the element inside the array (the first element is zero, the second element is one, and so on). An element of an associative array can be accessed using both an index number and an identifier. Regular arrays can be created with the "array" (page 784) operator. Associative arrays can be created with the hash (page 796) operator. The following examples demonstrate the creation of arrays and hashes.

Example 1: Array of numbers

```
{array( 2, 4, 8, 16 )}
```

This example creates an array containing four numbers. The array will consist of the following elements:

Index	Value of element
	2
1	4
2	8
3	16

Example 2: Array of strings

```
{array( 'This', 'is', 'a', 'test' )}
```

This example creates an array containing four strings. The array will consist of the following elements:

Index	Value
	'This'
1	'is'
2	'a'
3	'test'

Example 3: Associative array

```
{hash( 'Red', 16, 'Green', 24, 'Blue', 32 )}
```

This example creates an associative array containing three key-value pairs. The array will consist of the following elements:

Index	Key	Value
	Red	16
1	Green	24
2	Blue	32

Objects

Template objects are created by PHP code or by special template operators. An object consists of attributes where each attribute can be a different type. An attribute may represent any type of data (number, string, etc.), including another object. The contents of an attribute may be accessed by using the attribute's identifier. The system uses objects to represent complex data structures. For example, objects are used to represent information about content nodes, attributes, translations, roles, policies, etc. Please refer to the "Objects" (page 696) section of the "Reference" chapter for a complete overview of the available objects and their attributes. The following illustration shows the structure (with example values) of an object ("ezdate" (page 728)) that contains information about a date.

(see figure 3.6)

Attribute	Type	Value
timestamp	string	567990000
is_valid	boolean	TRUE
year	string	1988
month	string	01
day	string	01

Figure 3.6: The structure of the "ezdate" object.

3.3.3 Variable usage

Template variables must be referenced using dollar (\$) notation, for example: \$my_variable, \$object_array, etc. An eZ publish template variable is case sensitive. In other words, \$lollipop is not the same variable as \$LolliPop. Template variables can be created by the system (from PHP code) or by the author of the template (from within template code). Regardless of where a variable was created, it can be changed using the "set" (page 1025) function. Some templates have preset variables, for example, the main template (pagelayout) provides access to a collection of variables (page 181).

Creating and destroying variables

All variables used in a template must be declared and defined by the "def" (page 1020) function (short for define) before they can be used. A variable exists until the "undef" (page 1029) function (short for undefined) is used in order to destroy it. A previously declared variable will be automatically destroyed at the end of the template file in which it was created. The following example demonstrates the most basic use of the "def" and "undef" functions.

```
{def $temperature=32}

    {$temperature}

{undef}
```

The output of the example will be "32". After the {undef} function is called, the \$temperature variable will not be available. Both the "def" and the "undef" function can be used with multiple variables at the same time. In addition, the "undef" function can be used without any parameters. When called without parameters the "undef" function automatically destroys all variables that were previously created within the template. The following example demonstrates how the "def" and "undef" functions can be used to create and destroy multiple variables at the same time.

```
{def $weather='warm' $celsius=32 $fahrenheit=90}

The weather is {$weather}: {$celsius} C / {$fahrenheit} F <br />

{undef $celsius $fahrenheit}

The weather is still {$weather}. <br />

{undef}
```

The output of this example will be:

The weather is warm: 32 C / 90 F
The weather is still warm.

In the example above, the "def" function is used to create three new variables: \$temperature, \$celsius and \$fahrenheit. The "undef" function is used twice. The first time, it is used to destroy the \$celsius and \$fahrenheit variables. The second time it is called without parameters and thus the remaining variables (in this case only \$temperature) will be destroyed. For more information, please refer to the documentation page of the "def" (page [1020](#)) and "undef" (page [1029](#)) functions.

Changing the contents of variables

The contents/value of a variable can be changed at any time using the "set" (page [1025](#)) function. Please note that this function can be used to change the value of any variable, regardless of if it was created by the system or inside a template. No warning will be given if a system variable is changed. The "set" function can be used to change the value of any variable regardless of the variable's current type and the type of the new value. In other words, this function is capable of changing the type of a variable. The "set" function can not be used to change the value of an element/attribute of an array, hash or an object. In fact, the elements/attributes of arrays, hashes and objects can not be changed from within template code. The following example demonstrates the usage of the "set" function.

```
{def $weather='warm'}

The weather is {$weather}. <br />

{set $weather='cold'}

The weather is {$weather}.

{undef}
```

The output of the example will be:

The weather is warm.
The weather is cold.

Just like the "def" and "undef" functions, the "set" function can work with multiple variables at the same time. For more information, please refer to the documentation page of the "set" (page [1025](#)) function.

Accessing array elements

The elements of a simple/vector array can only be accessed using numerical indexes. This method is called "index lookup". The elements of an associative array can be accessed by using the key identifiers. This method is called "identifier lookup". In addition, the elements of an associative array can also be accessed using index values. The following example demonstrates the different lookup methods.

Index lookup

Index lookup is carried out by appending a period/dot and an index number to the name of a simple/vector or associative array. Index lookup may also be carried out by appending a matching pair of brackets that encapsulate the desired index value. The following example demonstrates how to access the elements of a simple array using index lookup. Please note the different syntaxes (dot and brackets).

```
{def $sentence=array( 'Life', 'is', 'very', 'good!' )}

The 1st element is: {$sentence.0} <br />
The 2nd element is: {$sentence.1} <br />
The 3rd element is: {$sentence[2]} <br />
The 4th element is: {$sentence[3]} <br />

{undef}
```

The code above will output the following:

```
The 1st element is: Life
The 2nd element is: is
The 3rd element is: very
The 4th element is: good!
```

Identifier lookup

Identifier lookup can be carried out by appending a period/dot and an identifier name to the name of an associative array. Identifier lookup may also be carried out by appending a matching pair of brackets that encapsulate the desired index value. The following example demonstrates how to access the elements of an associative array using the identifier lookup method. Notice the different syntax (use of dot and brackets).

```
{def $sentence=hash(
    'first', 'Life',
    'second', 'is',
    'third', 'very',
```

```
'fourth', 'good!' )}

The 1st element is: {$sentence.first} <br />
The 2nd element is: {$sentence.second} <br />
The 3rd element is: {$sentence[third]} <br />
The 4th element is: {$sentence[fourth]} <br />

{undef}
```

The following output will be produced:

```
The 1st element is: Life
The 2nd element is: is
The 3rd element is: very
The 4th element is: good!
```

Index lookup and associative arrays

As pointed out earlier, the elements of an associative array may also be accessed using the index method. The following example demonstrates this.

```
{def $sentence=hash( 'first', 'Life',
                     'second', 'is',
                     'third', 'very',
                     'fourth', 'good!' )}

The 4th element is: {$sentence.3} <br />
The 3rd element is: {$sentence.2} <br />
The 2nd element is: {$sentence[1]} <br />
The 1st element is: {$sentence[0]} <br />

{undef}
```

The following output will be produced:

```
The 4th element is: good!
The 3rd element is: very
The 2nd element is: is
The 1st element is: Life
```

Accessing object attributes

The attributes of an object can only be accessed using the attributes' identifiers. An identifier is just the name of an attribute (similar to the keys of an associative array). The following example demonstrates how the different attributes of a node object can be accessed from within a template.

```
The ID of the node: {$node.node_id} <br />  
The ID of the object encapsulated by the node: {$node.object.id} <br />  
The name of the object: {$node.object.name} <br />  
First time the object was published: {$node.object.published|l10n( shortdate  
)} <br />  
>
```

If the \$node variable contains a node that has ID number 44 and encapsulates object number 13 named "Birthday" published on the first of April in 2003, the following output will be produced:

```
The ID of the node: 44  
The ID of the object encapsulated by the node: 13  
The name of the object: Birthday  
First time the object was published: 01/04/2003
```

3.3.4 Array and object inspection

By using the "attribute" (page 892) template operator, it is possible to quickly inspect the contents of arrays and template objects. The operator creates an overview of available keys, attribute names and/or methods in an object or an array. By default, only the array keys and object attribute names (also called identifiers) are shown. By passing "show" as the first parameter, the operator will also display the values. The second parameter can be used to control the number of levels/children that will be explored (the default setting is 2). The following example demonstrates how the operator can be used to inspect the contents of an "ezcontentobjecttreenode" (page 721) object.

```
{$node|attribute( show, 1 )}
```

The following output will be produced:

Attribute	Type	Value
node_id	string	2
parent_node_id	string	1
main_node_id	string	2
contentobject_id	string	1
contentobject_version	string	10
contentobject_is_published	string	1
depth	string	1
sort_field	string	8
sort_order	string	1
priority	string	0
modified_subnode	string	1108118324
path_string	string	'/1/2/'
path_identification_string	string	"
is_hidden	string	0
is_invisible	string	0
name	string	'eZ publish'
data_map	array	Array(6)
object	object[ezcontentobject]	Object
subtree	array	Array(114)
children	array	Array(44)
children_count	string	44
contentobject_version_object	object[ezcontentobjectversion]	Object
sort_array	array	Array(1)
can_read	boolean	true
can_create	boolean	false
can_edit	boolean	false
can_hide	boolean	false
can_remove	boolean	false
can_move	boolean	false
creator	object[ezcontentobject]	Object
path	array	Array(0)
path_array	array	Array(2)
parent	object[ezcontentobjecttreenode]	Object
url	string	"
url_alias	string	"
class_identifier	string	'folder'
class_name	string	'Folder'
hidden_invisible_string	string	'-/-'
hidden_status_string	string	'Visible'

As the output shows, there is a lot of information that can be extracted from a node object. In addition to strings and numbers the object also consists of other objects. For example, the creator of the node is a "ezcontentobject" (page 712) object. The creator object can be further inspected

by doing the following:

```
{$node.creator|attribute( show, 1 )}
```

The following output will be produced:

Attribute	Type	Value
id	string	14
section_id	string	2
owner_id	string	14
contentclass_id	string	4
name	string	'Administrator User'
is_published	string	0
published	string	1033920830
modified	string	1033920830
current_version	string	1
status	string	1
current	object[ezcontentobjectversion]	Object
versions	array	Array(1)
author_array	array	Array(1)
class_name	string	'User'
content_class	object[ezcontentclass]	Object
contentobject_attributes	array	Array(5)
owner	object[ezcontentobject]	Object
related_contentobject_array	array	Array(0)
related_contentobject_count	string	0
reverse_related_contentobject_array	array	Array(0)
reverse_related_	string	0

contentobject_count		
can_read	boolean	false
can_create	boolean	false
can_create_class_list	array	Array(0)
can_edit	boolean	false
can_translate	boolean	false
can_remove	boolean	false
can_move	boolean	false
data_map	array	Array(5)
main_parent_node_id	string	13
assigned_nodes	array	Array(1)
parent_nodes	array	Array(1)
main_node_id	string	15
main_node	object[ezcontentobjecttreenode]Object	
default_language	string	'eng-GB'
content_action_list	boolean	false
class_identifier	string	'user'
class_group_id_list	array	Array(1)
name	string	'Administrator User'
matchingroup_id_list	boolean	false

Again, this object consists of a lot of information. As mentioned above, the "attribute" (page 892) operator can be used on both objects and arrays. The following example demonstrates how to inspect the "data_map" array (which reveals the object's attributes) of the node's creator object.

```
{$node.creator.data_map|attribute( show, 1 )}
```

The following output will be produced:

Attribute	Type	Value
first_name	object[ezcontentobjectattribute]Object	
last_name	object[ezcontentobjectattribute]Object	
user_account	object[ezcontentobjectattribute]Object	
signature	object[ezcontentobjectattribute]Object	
image	object[ezcontentobjectattribute]Object	

3.3.5 Control structures

The eZ publish template language offers a selection of mechanisms that can be used to solve common programmatic issues like for example condition control, looping, etc. The following list shows an overview of the available mechanisms:

- IF-THEN-ELSE
- SWITCH
- WHILE
- DO...WHILE
- FOR
- FOREACH

IF-THEN-ELSE

The IF (page 1052) construct allows for conditional execution of code fragments. It is one of the most important features of many programming languages. The eZ publish implementation makes it possible to do conditional branching by the way of the following elements: IF, ELSE and ELSEIF. The ELSE and ELSEIF elements are optional. The following examples demonstrate the use of this construct.

Example 1

```
{if eq( $var, 128 )}  
    Hello world <br />  
{else}  
    No world here, move along. <br />  
{/if}
```

Example 2

```
{if eq( $fruit, 'apples' )}  
    Apples <br />  
{elseif eq( $fruit, 'oranges' )}  
    Oranges <br />  
{else}  
    Bananas <br />  
{/if}
```

SWITCH

The SWITCH (page 1054) mechanism is similar to a series of IF statements used on the same expression. This construct is typically useful when the same variable needs to be compared to different values. It executes a piece of code depending on which value that matched a given criteria. The following example demonstrates basic use of this construct.

```
{switch match=$fruits}

{case match='apples'}
    Apples <br />
{/case}

{case match='oranges'}
    Oranges <br />
{/case}

{case}
    Unidentified fruit! <br />
{/case}

{/switch}
```

If the value of the \$fruits variable is "oranges", the following output will be produced:

Oranges

WHILE

The WHILE (page 1064) construct is the simplest loop mechanism that the template language offers. It tells eZ publish to execute the nested statement(s) repeatedly, as long as a given expression evaluates to TRUE. The value of the expression is checked for every loop iteration (at the beginning of the iteration). If the given expression evaluates to FALSE from the very beginning, the nested statement(s) will not be executed. The following example demonstrates basic use of this construct.

```
{while ne( $counter, 8 )}

    Print this line eight times {$counter} <br />
    {set $counter=inc( $counter )}

{/while}
```

If the initial value of \$counter is zero, the following output will be produced:

```
Print this line eight times (0)
Print this line eight times (1)
Print this line eight times (2)
Print this line eight times (3)
Print this line eight times (4)
Print this line eight times (5)
Print this line eight times (6)
Print this line eight times (7)
```

DO...WHILE

A DO...WHILE (page 1060) loop is very similar to WHILE loops, except that the expression is checked at the end of each iteration instead of in the beginning. The main difference is that this construct will always execute the first iteration (regardless of how the test expression evaluates). The following example demonstrates basic use of this construct.

```
{do}
    Keep printing this line {$counter} <br />
    {set $counter=inc( $counter )}

{/do while ne( $counter, 8 )}
```

3

If the initial value of \$counter is 0, the following output will be produced:

```
Keep printing this line (0)
Keep printing this line (1)
Keep printing this line (2)
Keep printing this line (3)
Keep printing this line (4)
Keep printing this line (5)
Keep printing this line (6)
Keep printing this line (7)
Keep printing this line (8)
```

FOR

Generic looping may be achieved using FOR (page 1061) loops. This construct supports looping over numerical ranges in both directions. In addition it also supports breaking, continual and skipping. The following example demonstrates basic use of this construct.

```
{for 0 to 7 as $counter}

    Value of counter: {$counter} <br />

{/for}
```

The following output will be produced:

```
Value of counter: 0
Value of counter: 1
Value of counter: 2
Value of counter: 3
Value of counter: 4
Value of counter: 5
Value of counter: 6
Value of counter: 7
```

3

FOREACH

The FOREACH (page 1062) construct can be used to iterate over arrays in different ways. The loop can be tweaked using miscellaneous techniques. The following example demonstrates basic use of this construct.

```
{foreach $objects as $object}

    {$object.name} <br />

{/foreach}
```

The example above will print out the names of the objects that are stored in the \$objects array. If this array stores 4 objects with the following names: "Emmett Brown", "Marty McFly", "Lorraine Baines" and "Biff Tannen", the following output will be produced:

```
Emmett Brown
Marty McFly
Lorraine Baines
Biff Tannen
```

3.3.6 Functions and operators

The eZ publish template language offers a collection of various functions (page 998) and operators (page 780) that can be used to carry out different tasks. In addition, it is possible to extend the system by creating custom operators for special needs. Custom operators have to be programmed in PHP.

Template functions

A function takes a set of named parameters, carries out a specific task and returns a result. It can be called anywhere in a template using the following syntax:

```
{function_name parameter1=value1 parameter2=value2 ...}
```

A function may take none, one or several parameters. The parameters must be specified after the function name, separated by spaces. Since each parameter is specified using the parameter's name, the parameters can be provided in any order. Each parameter must be assigned a value using the equal sign. The following illustration shows the typical usage of a commonly used function.

(see figure 3.7)

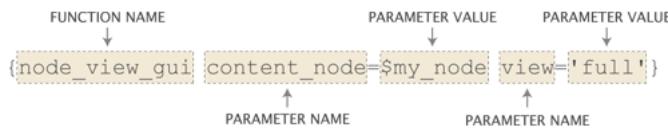


Figure 3.7: Typical components of a function call.

The example above calls the "node_view_gui" (page 1046) function. This function displays a node by including the template that is associated with the view mode. The node is specified using the "content_node" parameter. The desired view mode is specified using the "view" parameter.

Template operators

An operator takes unnamed parameters, carries out a specific task and returns a result. In addition, an operator is capable of handling a parameter which is passed to it using a pipe. It can be called anywhere in a template using the following syntax:

```
{$input_parameter|operator_name( parameter1, parameter2 ... )}
```

Because the operator only takes unnamed parameters, the parameters must be specified in the order dictated by the operator's documentation page. In addition, the parameters must be separated by commas. The following illustration shows the typical usage of a commonly used operator.

(see figure 3.8)

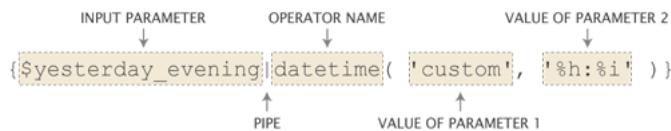


Figure 3.8: Typical components of a template operator call.

The example above demonstrates the usage of the "datetime" (page 816) operator. This operator can be used to convert a UNIX timestamp to a human readable format. The timestamp is provided by the `$yesterday_evening` variable as the input parameter. The first parameter tells the operator that the output should be formatted using a custom schema. The schema is defined by the second parameter (hours : minutes).

Piping

An operator takes input on the left hand side and produces output on the right hand side. A collection of operators can be glued together using pipes. A pipe makes sure that the output from one operator is presented as the input parameter to another operator. The following example demonstrates how pipes and operators can be used to create a string.

```
{concat( 'To ', 'The ')|prepend( 'Back ')|append( 'Future' )}
```

The following output will be produced:

```
Back To The Future
```

3.4 Basic template tasks

This section sheds light on some common issues related to template development.

Template inclusion

A template file can be included using the "include" (page 1012) function. Since this function makes it possible to include any file from any location within the eZ publish directory, it must be told that it should look for the file within the design directory. This can be done by prefixing the path/filename with "design:". The following example demonstrates how the include function can be used to include a template file called "footer.tpl", which is located in the templates directory of a design.

```
{include uri='design:footer.tpl'}
```

If the requested file is not found within the main design of the siteaccess, the system will search for it in the additional designs and the standard design. Please refer to the documentation of the automatic fallback system for more information about this feature.

Output washing

Variables that may contain bogus strings should always be washed using the "wash" (page 955) operator. This operator makes sure that the output does not contain any elements that may mess up the HTML generated by eZ publish. The following example demonstrates how the wash operator works.

```
{def $bogus_string='hello < world'}
{$bogus_string|wash()}
```

The following output will be produced:

```
hello < world
```

E-mail address obfuscation

In addition to securing proper output, the wash operator can also be used to obfuscate E-mail addresses on a web page. An obfuscated E-mail address has a less chance of getting picked up by a robot searching for E-mail addresses to put on a spammer's list. The following example demonstrates how the wash operator can be used with an E-mail address.

```
{def $email_address='allman@example.com'}
{$email_address|wash( 'email' )}
```

The following output will be produced:

```
allman[at]example[dot]com
```

String concatenation

The "concat" (page 924) operator makes it possible to glue several strings together in order to produce a single string. The following example demonstrates how this operator works.

```
{def $my_string='sausage'}
{concat( 'Liver ', $my_string, ' sandwitch' )}
```

The following output will be produced:

```
Liver sausage sandwitch
```

Custom view parameters

The URL of a node view request may contain custom parameters. The custom view parameters must be specified at the very end of the URL using a special notation. For each parameter, a name and a value must be specified. The name must be encapsulated by parenthesis. Each element must be separated by slashes. The following example demonstrates how custom parameters can be used (in addition to the view parameters) in a system URL that requests a node.

```
http://www.example.com/content/view/full/13/(color)/green/(amount)/34
```

The same parameters can be appended to the virtual URL of the node:

```
http://www.example.com/company/about_us/(color)/green/(amount)/34
```

When custom view parameters are used, the system will create an associative array using the name of the provided parameters as the keys. All parameter values will be treated as strings. The array will be represented by the \$view_parameters variable in the template. The parameters given in the examples above will produce an associative array with the following contents:

Key	Type	Value
color	string	green
amount	string	34

The following example demonstrates how the custom view parameters can be accessed in the template that is used to display the node.

```
The color is: {$view_parameters.color} <br />
The amount is: {$view_parameters.amount} <br />
```

The following output will be produced:

```
The color is: green
The amount is: 34
```

3.4.1 URL handling

Whenever a link, a non-content specific image, a stylesheet, etc. is to be included, a suitable template operator must be used in order to ensure that the path to the included file is correct. At any time, one of the following operators should be used:

- ezurl
- ezimage
- ezdesign

ezurl

The "ezurl" (page 966) operator makes sure that a URL works regardless of the location of the eZ publish folder, the access method (page 146) and the environment that eZ publish is running in (non virtual host, virtual host (page 70), etc.). It is only the eZ publish specific part of the URL that needs to be provided. The rest (http://, host, domain, directory, siteaccess, port, etc.) will be generated by the operator. The final output will be a valid address. This approach makes it possible to use generic URLs in template without the risk of having to modify every address when the site is moved and/or when the access method is changed. By default, the "ezurl" operator outputs an address that is already encapsulated by two double quotes. In other words, the output can be fed directly to an hyperlink reference in the HTML code. The following examples demonstrate the usage of this operator.

Link to a module/view (using a system URL)

```
<a href={'/user/login'|ezurl()}>Login</a>
```

The example above demonstrates how to create a link to the *login* view of the *user* module. The "/user/login" is just an example, another example would be a link to a node: "/content/view/full/34". If eZ publish is running in a directory called "ezpublish" on www.example.com using the URL access method and the name of the siteaccess is "my_company", the operator will produce the following output:

```
"http://www.example.com/ezpublish/index.php/my_company/user/login"
```

If eZ publish is running in a virtual host mode (page 70) and uses the host access method, the following URL will be produced:

```
"http://www.example.com/user/login"
```

Link to a node (using the node's virtual URL)

When a link to a node (using the node's virtual URL, also known as URL alias) is created, the address must be piped through the "ezurl" operator. The reason for this is that the internal URL table only contains the eZ publish specific part of the URLs. The following example demonstrates how to use the "ezurl" operator to create a valid virtual URL for a node.

```
<a href={$node.url_alias|ezurl()}>Link to a node</a>
```

If the URL alias of the node is "company/about_us" and eZ publish is running in a virtual host environment using the host access method, the following URL will be produced:

"http://www.example.com/company/about_us"

For information about how eZ publish treats URLs, please refer to the "URL translation" (page 152) section of the "Concepts and basics" chapter.

ezimage

The "ezimage" (page 963) operator works in the same way as the "ezurl" operator (described above), except that it does not include the "index.php" part. This operator must be used every time a non content specific image is included in a template. The image must be placed in the "images" directory of one of the designs that are used by the siteaccess. The operator produces a valid link to the image regardless of the directory, access method and/or the environment that eZ publish is running in. The following example demonstrates how the "ezimage" operator should be used.

```
<img src='women.jpg'|ezimage() alt="This is my image." ... />
```

If eZ publish is using the host access method and the siteaccess is using a design called "my_design", the operator will produce the following output:

"http://www.example.com/design/my_design/images/women.jpg"

If the image is placed inside a subdirectory within the "images" directory, the name of the subdirectory must be specified in the template. If the requested file is not found within the main design of the siteaccess, the system will search for it in the additional designs and the standard design. Please refer to the documentation of the automatic fallback system for more information about this feature.

ezdesign

The "ezdesign" (page 962) operator works in the same way as the "ezurl" operator (described above), except that it does not include the "index.php" part. This operator must be used every time a design element (style sheets, Javascript, etc.) is included in a template. The operator takes care of producing a valid link for the given design component by providing the root to the design directory which contains the target file. The following example demonstrates the proper way of including a CSS file using this operator.

```
...
<style type="text/css">
    @import url({'stylesheets/my_stuff.css'|ezdesign()});
</style>
...
```

If eZ publish is using the host access method and the siteaccess is using a design called "my_design", the operator will produce the following output:

```
"http://www.example.com/design/my_design/stylesheets/my_stuff.css"
```

If the requested file is not found within the main design of the siteaccess, the system will search for it in the additional designs and the standard design. Please refer to the documentation of the automatic fallback system for more information about this feature.

3.5 Information extraction

Information that is stored by eZ publish can be extracted using the "fetch" (page 813) template operator. This operator gives access to the fetch functions that a module provides. It is typically used to extract nodes, objects, etc. using the content module. The fetch operator can only be used with modules that provide support for data fetching. Please refer to the "Fetch functions" (page 1088) section of the reference chapter for a complete overview of the fetch functions. The following model and table shows the usage and the parameters of the fetch operator.

```
fetch( <module>, <function>, <parameters> )
```

Parameter	Description
module	The name of the target module.
function	The name of the fetch function within the target module.
parameters	An associative array containing the function parameters.

A module's fetch functions and parameters are defined in the "function_definition.php" file within the directory of the module.

Fetching a single node

The following example demonstrates how the fetch operator can be used to extract a single node from the database.

```
{def $my_node=fetch( content, node, hash( node_id, 13 ) )}  
...  
{undef}
```

The example above instructs eZ publish to fetch a single *node* from the *content* module. Only one parameter is given, which is the ID number of the node that should be fetched. The operator will return an "ezcontentobjecttreenode" (page 721) object which will be stored in the \$my_node variable. This variable can then be used to extract information about the node and the object that it encapsulates. For example, it is possible to extract the name, attributes and the time when the object was published. If the node is unavailable / non-existing or the currently logged in user doesn't have read access to it, the operator will return a FALSE boolean value.

Fetching multiple nodes

It is possible to fetch all the nodes that are directly below a specific node. This can be done by using *list* instead of *node* as the second parameter to the "fetch" operator. The following example

demonstrates how the fetch operator can be used to extract all the nodes that are directly below node number 13.

```
{def $my_node=fetch( content, list, hash( parent_node_id, 13 ) )}  
...  
{undef}
```

The operator will return an array of "ezcontentobjecttreenode" (page 721) objects. The list fetch function of the content module can take several parameters. These parameters are optional and can be used to finetune the fetch for example by filtering out specific nodes. The following table gives an overview of the most commonly used parameters.

Parameter	Description
sort_by	The method and direction that should be used when the nodes are sorted (must be specified as an array).
limit	The number of nodes that should be fetched.
offset	The offset at which the fetch should start.
class_filter_type	The type of filter that should be used, either "include" or "exclude".
class_filter_array	The type of nodes that should be included or excluded by the filter (must be specified as an array).

The following example demonstrates how to fetch an alphabetically sorted array of the ten latest articles that are directly below node number 13.

```
{def $my_node=fetch( content,  
                      list,  
                      hash( parent_node_id,      13,  
                            limit,            10,  
                            class_filter_type, include,  
                            class_filter_array, array( 'article' ) ) )}  
...  
{undef}
```

Please refer to the documentation page of the "list" (page 436) fetch function for a complete overview of the available parameters and examples of usage.

3.5.1 Outputting node and object data

Once an "ezcontentobjecttreenode" (page 721) object representing a node is available in a template variable, it can be used to output information about the node and the contents of the object that the node encapsulates. The following text demonstrates the extraction of the most common elements.

General information

The name of the object

```
{$node.name|wash}
```

The name of the object is directly available through the node (in other words it is possible to reach it by \$node.name instead of \$node.object.name). The "wash" (page 955) operator is used for making sure that the output doesn't contain any bogus characters and/or sequences that may mess up the HTML.

The date/time when the object was first published

```
{$node.object.published|l10n( 'shortdatetime' )}
```

Since the publishing value is stored as a UNIX timestamp, it must be properly formatted for output. This can be done by using the "l10n" (page 821) operator, which makes it possible to format different types of values according to the current locale settings.

The date/time when the object was last modified

```
{$node.object.modified|l10n( 'shortdatetime' )}
```

Since the modification value is stored as a UNIX timestamp, it must be properly formatted for output. This can be done by using the "l10n" (page 821) operator, which makes it possible to format different types of values according to the current locale settings.

The name of the user who initially created the object

```
{$node.object.owner.name|wash}
```

The name of the user who last modified the object

```
{$node.object.current.creator.name|wash()}
```

The name of the class which the object is an instance of

```
{$node.object.class_name|wash()}
```

Object attributes

The attributes of the object can be reached by the way of the "data_map" method. This method returns an associative array of "ezcontentobjectattribute" (page 717) objects where each object represents one of the attributes. The keys of the array are the class attribute identifiers. The following example demonstrates how an attribute called "first_name" can be reached using the object's data map.

```
{$node.object.data_map.first_name}
```

The example above will not produce any valuable output because the requested data needs to be formatted. There are two ways of outputting the contents of attributes:

- Raw output (the ".output" extension)
- Formatted output (the "attribute_view_gui" function)

The main difference between raw and formatted output is that formatted output makes use of a template which in turn outputs the requested data. Raw output simply outputs the data within the same template where the request for output was issued. Output should always be presented through the "attribute_view_gui" (page 1035) function. The raw output method should only be used when/if necessary (for example when checking the value of an attribute using an IF statement).

Raw output

Raw output is exactly what the definition indicates: a raw dump of the contents that are stored by the attribute. The actual syntax depends on the datatype that represents the attribute. In most cases, it is possible to generate the output by appending ".output" to the identifier.

Generic solution

The following example demonstrates how to output the contents of an attribute called "my_attribute".

```
{$node.object.data_map.my_attribute.content}
```

XML block

The following example demonstrates how to output the contents of an XML block called "my_xml".

```
{$node.object.data_map.my_xml.content.output.output_text}
```

Image

The following example demonstrates how to output an image stored by an attribute called "my_image".

```

```

Formatted output

Each datatype has a set of templates which are used to display the contents in different contexts. There are at least two templates for each datatype: a view template and an edit template. While the view template is used to display information, the edit template is used when the data is being edited. The default templates for the datatypes are located within the standard design: "/design/standard/templates/content/datatype".

The "attribute_view_gui" (page 1035) function makes it possible to display the contents of an attribute by inserting the view template of the datatype that the attribute uses. The following example demonstrates how this function can be used.

```
{attribute_view_gui attribute=$node.object.data_map.name_of_any_attribute}
```

The example above will generate proper output for any attribute (regardless of the datatype).

3.6 The template override system

The template override system makes it possible to use other templates than the default ones (specified in the code for the different views and templates). This mechanism allows the creation of template overrides for virtually any template that is used by eZ publish (including templates that are requested by the "include" (page 1012) template function using the "design:" prefix). In particular, template overrides are typically useful for displaying different types of nodes in different ways.

An override for a view template is usually activated by a set of conditions. If the conditions match, the alternate template will be used. Different views provide different conditions, some views do not provide any conditions at all. Please refer to the "Template override conditions" (page 1065) section of the "Reference" chapter for a complete overview of the available match rules. The most flexible set of conditions are provided by the "view" view of the "content" module (used when a node is displayed). The following illustration shows how the override mechanism plugs into the rest of the system.

(see figure 3.9)

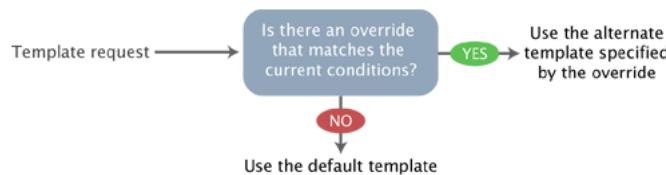


Figure 3.9: *The override system.*

The template overrides must be defined in the "override.ini.append.php" file of a siteaccess. This file consists of override blocks. A block is a named set of rules that tells eZ publish to use an alternate template in a specific situation. For each block, the following information must be specified:

- A unique name for the override.
- The template that should be overridden.
- The template that should be used instead of the one being overridden.
- The name of the directory in which the override template resides (usually "templates").
- A set of conditions/rules that control when the override should be activated.

Please note that the rules/conditions are optional. If no rules are specified, the override will always be active. The following illustration shows a typical example of a template override with additional explanations.

(see figure 3.10)

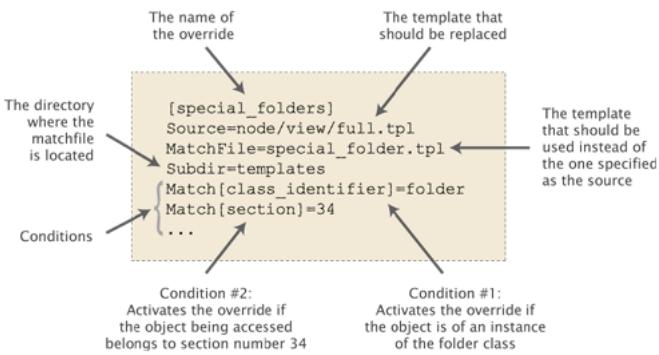


Figure 3.10: *Template override example.*

The example above defines an override called "special_folders". This override will be used when the system is requested to display a node using full view. The override will only be activated if the object referenced by the node is an instance of the folder class and if it belongs to section number 34. When the override is activated, the system will attempt to use the alternate template ("override/templates/special_folder.tpl", located in the main design). If eZ publish is unable to find the alternate template, it will look for it in the additional designs and the standard design. Please refer to the documentation page of the "Automatic fallback system" for more information about this feature.

3

Multiple / conflicting overrides

The priorities of the overrides are determined by their positions in the file. If there are several overrides with similar/equal rules, eZ publish will use the first override that matches and thus the rest of the overrides will be omitted. Because of this, overrides that are for example activated on a node ID or an object ID basis should always be placed first; otherwise they might never be triggered because of the presence of a more generic override with a higher priority.

3.6.1 Template override example

The following example demonstrates how the template override system can be used to display alternate templates in different situations.

Let's say that we have a simple content tree made up of two folders: "News" and "Products". The "News" folder contains news articles and the "Products" folder contains products. The following illustration shows an example of such a tree.

(see figure 3.11)

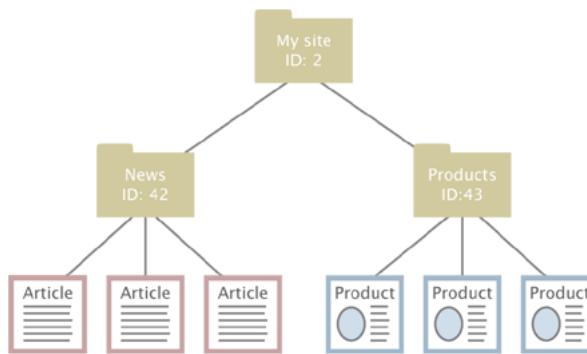


Figure 3.11: Example content node tree.

Without any overrides, eZ publish will most likely display all the nodes using the same template. This would probably be the default full view template located in the standard design. However, what if we wish to display custom/alternate templates for the different nodes? We would perhaps like the system to behave in the following way:

- Display a special "welcome" template when the "My site" node is accessed.
- Display a custom folder template when a folder is accessed.
- Display a custom article template when a news article is accessed.
- Display a custom product template when a product is accessed.

The requests in the list above can be easily achieved by creating a couple of overrides. The welcome page should be solved using an override that is triggered by the identification number of the "My site" node. The rest of the requests can be solved using the class identifier key, which allows an override to be triggered when an object of a certain class is accessed. The following example shows the contents of an "override.ini.append.php" file that makes this possible:

```
# Override for welcome page
[welcome_page]
Source=node/view/full.tpl
MatchFile=my_welcome.tpl
```

```

Subdir=templates
Match[node]=2

# Override for folders
[my_folder]
Source=node/view/full.tpl
MatchFile=my_folder.tpl
Subdir=templates
Match[class_identifier]=folder

# Override for articles
[news_articles]
Source=node/view/full.tpl
MatchFile=my_article.tpl
Subdir=templates
Match[class_identifier]=article

# Override for products
[products]
Source=node/view/full.tpl
MatchFile=my_product.tpl
Subdir=templates
Match[class_identifier]=product

```

The alternate templates should be placed in the "override/templates" subdirectory of the main design used by the siteaccess. The following illustration shows where the templates would be located in a design called "example".

(see figure 3.12)

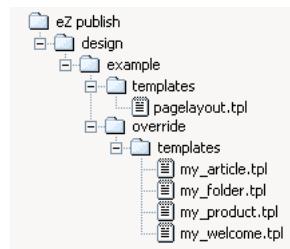
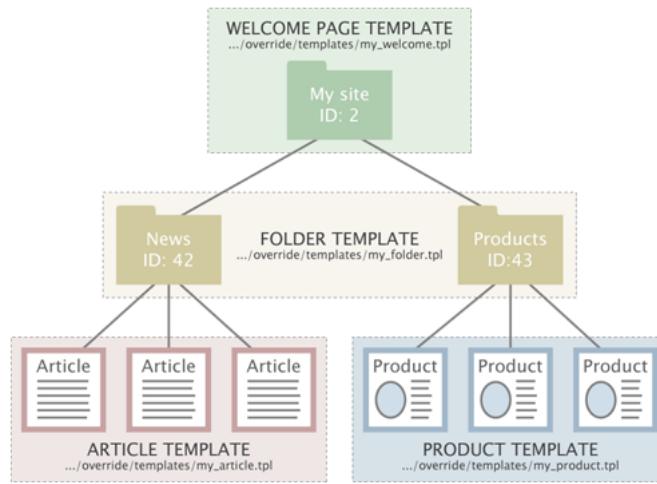


Figure 3.12: *Pagelayout + override templates in example design.*

When the system is in use, the different overrides would be activated based on the given conditions. The following illustration shows where/when the different alternate templates would be used.

(see figure 3.13)

Every time a node referencing a folder object is viewed, the system will use the "my_folder.tpl"

Figure 3.13: *Template override example.*

template. When an article is viewed, the "my_article.tpl" template will be used. When a product is viewed, the "my_product.tpl" template will be used. When node number 2 (the "My site" node) is viewed, the "my_welcome.tpl" will be used.

Chapter 4

Features

This chapter will contain information about miscellaneous eZ Publish features along with instructions revealing how to configure and use these features. Currently it only contains documentation about the built-in search engine, the notification system and the WebDAV server.

4.1 Notifications

eZ Publish has a built-in notification system that allows users to be informed about miscellaneous events that occur. It is possible to be notified when objects are updated or published, when workflows are executed and so on.

There are two built-in types of notifications:

- Subtree notifications
- Collaboration notifications

Subtree notifications

It is possible to subscribe for notifications about a subtree. For example, if you have a set of articles located under a folder called "Business", a user can subscribe for *subtree notifications* for this folder. The system will then send an E-mail to the user every time changes are made under the "Business" folder. The following changes will trigger a notification:

- When a new node is published within the subtree.
- When the contents of an existing node is changed.

A user can choose to receive notifications in the form of a single E-mail or as a digest of messages.

Collaboration notifications

The eZ Publish collaboration system allows you to work together with other people so that you can approve/reject any changes they made when it comes to content. For example, you can specify that all the changes made in the "Standard" section (page 136) can not be published without your approval. (This can be done by creating a new "Approve" event (page 773) within a new workflow (page 163) initiated by the "content-publish-before" trigger function.) If somebody (except you, the administrator) edits content located under the "Standard" section, the system will generate new collaboration messages. For example, if somebody changes article "A", the system will generate a new collaboration message "article A awaits your approval" for you and another collaboration message "article A awaits approval by editor" for the user who changed it.

To view your collaboration messages, click the "My Account" tab in the administration interface and then access the "Collaboration" link on the left. You will be able to review/approve/reject the changes.

You can use collaboration notifications to be notified by E-mail about new collaboration messages. The system will send you an E-mail every time a new collaboration message is generated for you.

Processing notifications

In the root of the eZ Publish directory there is a file called "runcronjobs.php". It takes care of processing the workflows, notifications and other tasks that should be processed in the background. If you are going to use the notification system, "runcronjobs.php" must be executed periodically. The most common way to do this is to set up a scheduled job that runs every 30-60 minutes or so. Please refer to "[The cronjob script](#)" and "[Cron jobs](#)" sections for more information.

In accordance with the instructions specified in the "cronjobs/notification.php" file, "runcronjobs.php" launches the main notification processing script "kernel/classes/notification/eznotificationeventfilter.php".

If you need to launch this script manually, add the "notification/runfilter" notation to the administration interface URL and then click the "Run notification filter" button there (see the next screenshot).

(*see figure 4.1*)



Figure 4.1: *The notification filter interface.*

Please note that processing notifications may cause a timeout error if there is a huge amount of notification events in the database. Because of this, the "runfilter" view of the "notification" module (page 546) should only be used for testing and debugging.

4.1.1 Using the admin interface

Subtree notifications

Subscribing

You can easily subscribe for subtree notifications about an object using either the context menu or the notification settings interface.

Using the context menu

To subscribe for subtree notification for an object, you should do the following:

1. Log in to the administration interface. You should see the "Content structure" tree on the left where the top node is selected. The following screenshot shows how the system will display the contents of the selected node and the list of its subitems. (*see figure 4.2*)
2. Locate the desired node in the "Content structure" tree or the "Sub items" window, click on its icon and select "Add to my notifications" from the context menu - this is shown in the screenshot below. (*see figure 4.3*)
3. The system will add a new subtree notification and show you a confirmation: (*see figure 4.4*)

Using the notification settings interface

It is possible to subscribe for subtree notifications for an object by adding this object to the "My item notifications" list located towards the bottom of the notification settings interface. The following text reveals how this can be done:

1. Click the "My Account" tab in the administration interface and select the "My notification settings" link on the left. You will be taken to the notification settings interface as shown in the screenshot below. (*see figure 4.5*)
This interface can be also accessed by adding the "/notification/settings" notation to the site URL.
2. Look at the "My item notifications" list located towards the bottom of the notification settings interface. All the items that you have already subscribed for are listed here. Click the "Add items" button to add a new notification.
3. The system will bring up the browse interface which will allow you to select the desired nodes: (*see figure 4.6*)

Use the list to select the node which encapsulates the object that you wish to be notified about. Please note that it is possible to select multiple nodes/objects at the same time. You can navigate the list by clicking on the names of the nodes. If the desired node is located

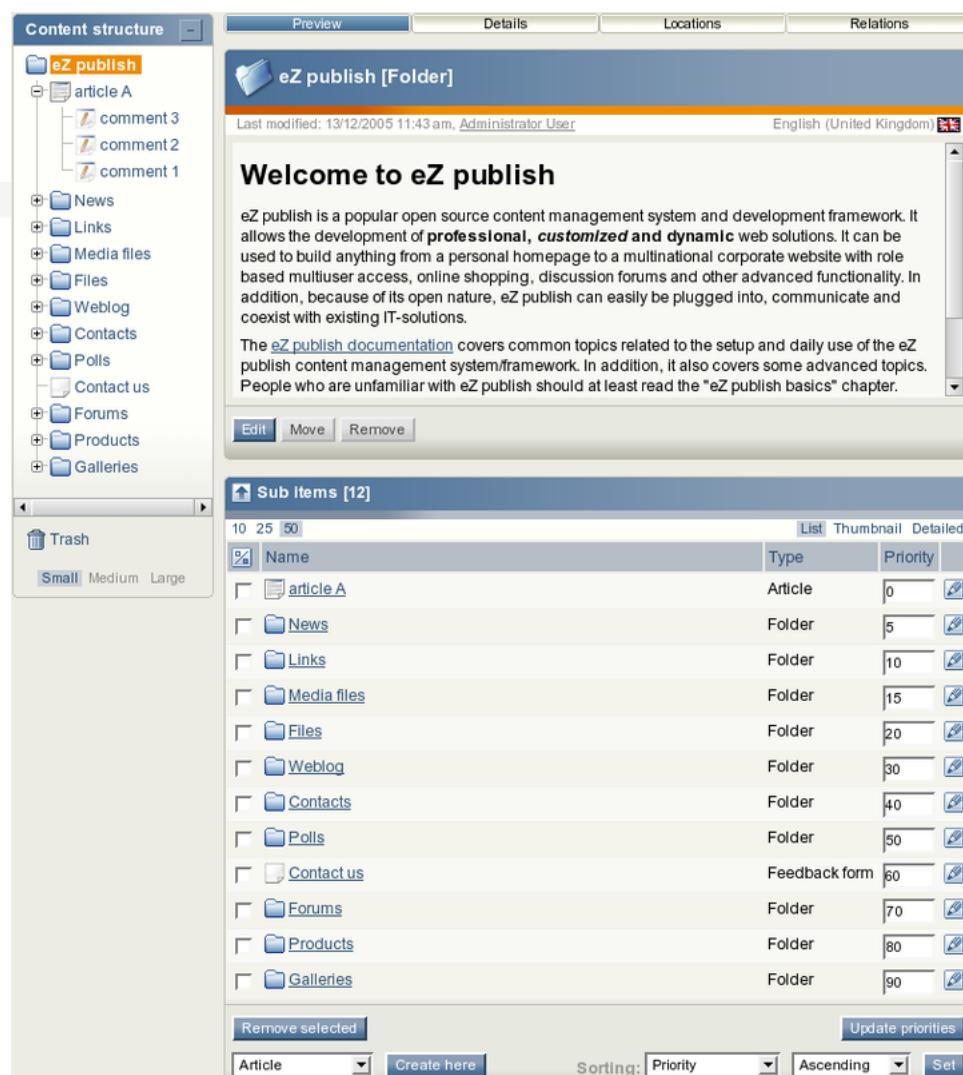


Figure 4.2: Browsing the content tree.

outside the "Content structure" tree then simply click the up arrow icon/button until it brings you to the root of the tree. This operation will allow you to for example switch to the "User accounts" tree and select user groups that are located there. The following illustration shows the up-arrow. (*see figure 4.7*)

It is possible to reconfigure how the list is displayed. For example, you can set the quantity of objects per page by clicking the "10" / "25" and "50" links. If you wish to browse image objects as thumbnails, simply click the "Thumbnail" button.

4. When you're finished selecting the desired object(s) (simply use the checkboxes to do this) click the "OK" button. The system will subscribe you for subtree notifications about these objects and add them to the "My item notifications" list.

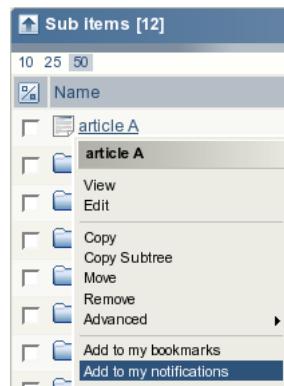


Figure 4.3: *Subscribing to subtree notifications using the context menu.*

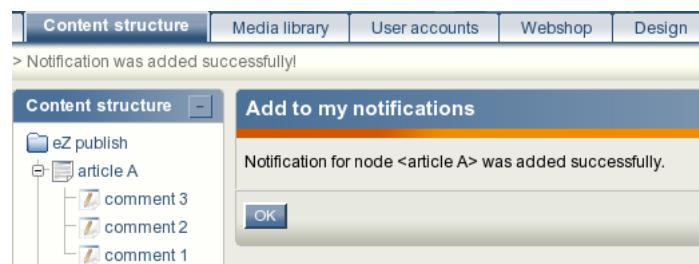


Figure 4.4: *The "notification added" confirmation for administrators.*

Setting up the digest mode

If you wish to receive the subtree notifications as a daily/weekly/monthly digest, enable the digest mode as described below.

1. Access the notification settings interface either by adding the "/notification/settings" notation to the URL or selecting "My Account - My notification settings" in the administration interface.
2. The digest settings are located at the top of the notification settings interface. By default, the digest mode is disabled (as shown in the screenshot below). (*see figure 4.8*)
To enable the digest mode, select the "Receive all messages combined in one digest" checkbox and choose how often the digest should be sent to you.
 - Once a day, at some fixed time (from 0:00 to 23:00).
 - Once a week, on some fixed day (from Sunday to Saturday).
 - Once a month, on some fixed day (from 1 to 31).
3. Click the "Apply changes" button to save your settings.

The screenshot shows the 'My notification settings' page. At the top, there are tabs for Content structure, Media library, User accounts, Webshop, Design, Setup, and My account. The 'My account' tab is selected. Below the tabs, a breadcrumb navigation shows 'Notification settings'. The main area is titled 'My notification settings'. It contains several sections:

- Receive all messages combined in one digest**: A checkbox is checked.
- Receive digests**: Three radio button options are available: 'Daily, at' (selected), 'Once per week, on' (Sunday), and 'Once per month, on day number' (1).
- Collaboration notification**: A checkbox for 'Approval' is checked.
- Apply changes**: A blue button.
- My Item notifications [1]**: A table with one row:

Name	Type	Section
article A	Article	Standard

 Buttons: 'Remove selected' and 'Add items'.

Figure 4.5: Notification settings for administrators.

Unsubscribing

If you no longer wish to receive notifications about an object, use the following instructions to unsubscribe.

1. Access the notification settings interface either by adding the "/notification/settings" notation to the URL or selecting "My Account" and then "My notification settings" in the administration interface.
2. The "My item notifications" list located towards the bottom of the notification settings interface contains all the items that you have already subscribed for. Use checkboxes to select the item(s) that you no longer wish to be notified about (see the screenshot below). (see figure 4.9)
3. Click the "Remove selected" button. The system will remove the selected item(s) from the list of notifications and thus you will no longer receive any messages about that/those object(s).

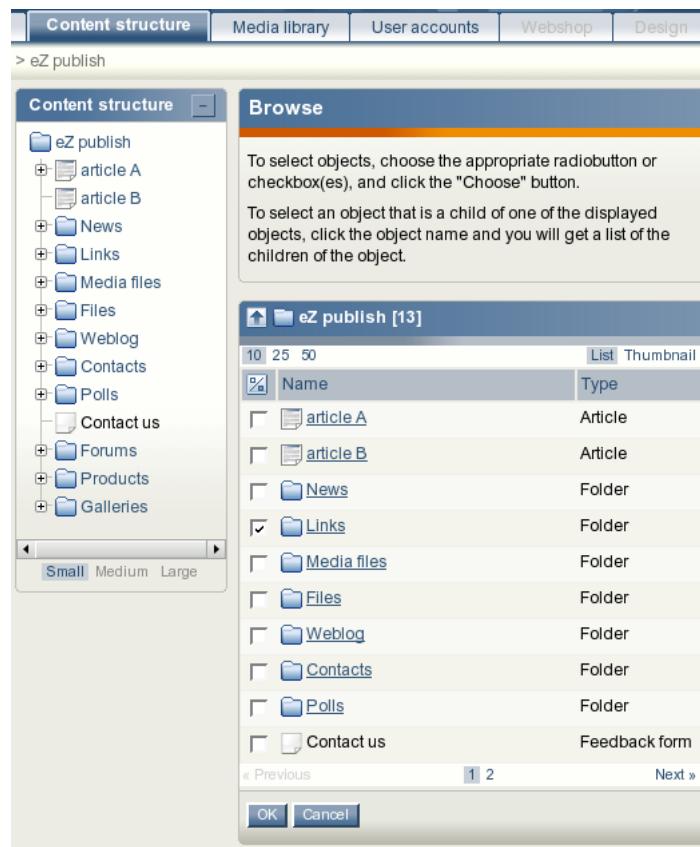


Figure 4.6: Browsing the content tree.



Figure 4.7: The "Up" button

Receive all messages combined in one digest

Receive digests

Daily, at

Once per week, on

Once per month, on day number

If day number is larger than the number of days within the current month, the last day of the current month will be used.

Figure 4.8: Digest settings

Collaboration notifications

If you're using the collaboration system to work together with other people, you may wish to be notified by E-mail every time a new collaboration message is created for you. In this case, you

My Item notifications [3]			
10	25	50	
%	Name	Type	Section
<input type="checkbox"/>	article A	Article	Standard
<input type="checkbox"/>	News	Folder	Standard
<input checked="" type="checkbox"/>	Links	Folder	Standard
Remove selected		Add items	

Figure 4.9: *The list of items for subtree notifications.*

should enable the collaboration notifications feature. The following text describes how to do this.

1. Access the notification settings interface either by adding the ”/notification/settings” notation to the URL or selecting ”My Account” and then ”My notification settings” in the administration interface.
2. Look at the ”Collaboration notification” section located under the digest settings. By default, the collaboration notifications are disabled. If you wish to receive collaboration notifications, select the ”Approval” checkbox as shown in the following screenshot. (*see figure 4.10*)

Collaboration notification

Choose which collaboration items you wish to get notifications for.

Approval

[Apply changes](#)

Figure 4.10: *Settings for collaboration notifications.*

3. Click the ”Apply changes” button to save your settings. The system will then send you an E-mail every time a new collaboration message is generated for you.

Please note that collaboration notifications do not support digest mode.

4.1.2 Using an actual site

Subtree notifications are available for those users who are allowed to use the "notification" module (page 546) by the role/policy settings. Please refer to "Granting access to notifications (page 241)" for more information about access rights. Collaboration notifications are only available when you're using the administration interface.

Subscribing for subtree notifications

A user can subscribe for subtree notifications about the item that is being viewed by clicking the "Keep me updated" button. The following screenshot shows a forum from one of the standard sites.

(see figure 4.11)

This is a demo forum where you can discuss small talk or other programming languages ;)

Topic	Replies	Author	Last reply
How big can a colour be?	1	21/04/2004 11:36 am Administrator User	21/04/2004 11:37 am Administrator User Not TRUE!

Figure 4.11: The "keep me updated" button.

After clicking this button, the system will add a new subtree notification and show a confirmation:

(see figure 4.12)

Notification was added successfully!

Add to my notifications

Notification for node <Small talk> was added successfully.

OK

Figure 4.12: The "notification added" confirmation for users.

In the standard sites, the "Keep me updated" button is always displayed on the forum pages while other pages do not contain this button. The forum is controlled by the following templates:

- design/your_siteaccess/override/templates/full/forum.tpl
- design/your_siteaccess/override/templates/full/forum_topic.tpl

Please refer to "Adding the "Keep me updated" button (page 239)" for more information about adding the update button to other templates/pages.

Setting the digest mode

The notification settings can be accessed regardless of the siteaccess/design that is used (as long as the permissions are ok). You can do the following to access the interface:

1. After logging in to the system, add the "/notification/settings" notation to the site URL (<http://www.example.com/notification/settings>) in order to access the notification settings. You should see the digest settings and the "Node notification" list (as shown in the following screenshot). (see figure 4.13)

The screenshot shows the 'Notification settings' page. At the top, there is a checkbox labeled 'Receive all messages combined in one digest'. Below it is a dropdown menu for 'Time of day' set to '0:00'. There are three radio button options: 'Daily', 'Weekly, day of week' (which has 'Sunday' selected), and 'Monthly, day of month' (which has '1' selected). A note below the monthly selection states: 'If the day of month number you have chosen is larger than the number of days in the current month, then the last day of the current month will be used instead.' Below this is a 'Node notification' section with a table:

Name	Class	Section	Select
News	Folder	1	<input type="checkbox"/>
Links	Folder	1	<input type="checkbox"/>

At the bottom are 'Remove', 'Store', and 'Cancel' buttons.

Figure 4.13: Notification settings for users.

2. By default, the digest mode is disabled. To enable the digest mode, select the "Receive all messages combined in one digest" checkbox, and choose how often the digest should be sent to you.
 - Once a day, at some fixed time (from 0:00 to 23:00).
 - Once a week, on some fixed day (from Sunday to Saturday).
 - Once a month, on some fixed day (from 1 to 31).
3. Click the "Store" button to save your settings. (If you wish to discard changes, simply click the "Cancel" button.)

Unsubscribing

If you no longer wish to receive subtree notifications about an object, follow these instructions to unsubscribe:

1. Access your notification settings by adding the ”/notification/settings” notation to the site URL (<http://www.example.com/notification/settings>).
2. The ”Node notification” list located under the digest settings contains all the items that you have already subscribed for (look at the previous screenshot). Use checkboxes to select the item(s) that you no longer wish to be notified about.
3. Click the ”Remove” button. The system will remove the selected item(s) from the list of notifications.

Please note that you can customize the notification settings template(s) by copying the default templates from either the standard or the admin design and changing them to suit your site.

4.1.3 Adding a "Keep me updated" button

A user can subscribe for subtree notifications for the page that is being viewed by making use of a "Keep me updated" button. Many of the default templates do not contain this button. The only exception is made for the forum pages where the button code is included into the following templates:

- design/your_siteaccess/override/templates/full/forum.tpl
- design/your_siteaccess/override/templates/full/forum_topic.tpl

Returning to the previous example, if you have a set of articles located under a folder called "Business", your users will not be able to subscribe for subtree notifications for this folder as long as there is no "Keep me updated" button there. Please note that the user must be logged in to make use of this feature.

You can easily add the "Keep me updated" button by inserting the following code into the override templates. For example, you can add this code to the "design/your_siteaccess/override/templates/full/folder.tpl" template:

```
<form method="post" action={'/content/action'|ezurl}>
<input type="hidden" name="ContentNodeID" value="{$node.node_id}" />
<input type="submit" name="ActionAddToNotification" value="Keep me updated" />
</form>
```

After clearing the caches, the "Keep me updated" button will appear every time a user is viewing a folder. The same changes can be easily done for your articles and other content objects.

Please note that some of the default templates may already contain a "/content/action" form. In this case, make sure that all the variables listed in the above code fragment are present inside this form in the template. You can also have several forms posting data to"/content/action".

If you wish to have the button present in the pagelayout then you'll have to do it a bit differently. The reason for this is that the \$node variable is not present in the pagelayout.

```
{* Check if we have a node... *}
{if $module_result.node_id}

<form method="post" action={'/content/action'|ezurl}>

<input type="hidden" name="ContentNodeID" value="{$module_result.node_id}" />
<input type="submit" name="ActionAddToNotification" value="Keep me updated" />

</form>

{/if}
```

4.1.4 Customizing the E-mails

It is possible to customize the notification E-mails by modifying templates. For example, the "plain.tpl" template located in the "templates/notification/handler/ezgeneraldigest/view/" directory of the standard design is the main notification template. It controls how the E-mails will be generated.

If you need to make changes to this template, you should copy it to your custom design and change it there. For example, you could copy it and add some additional/static text that will appear in all E-mails that are sent out. Remember to clear the caches before testing the changes. Please note that you should not change the default template but instead copy them to your own design.

4.1.5 Granting access to notifications

The built-in permission system controls whether users are allowed to use notifications or not. The following text explains how you can check and assign the necessary permissions.

Checking the access rights

The following text explains how you can view a user or a user group and check if the user or the group is allowed to access the "notification" module (page 546).

1. Log in to the administration interface and click the "User accounts" tab. You should see your users and groups on the left.
2. Select the target user/group using the tree or the "Sub items" window. (*see figure 4.14*)

The screenshot shows the TYPO3 Backend User Accounts interface. The top navigation bar includes tabs for Content structure, Media library, User accounts (which is active), Webshop, Design, Setup, and My account. Below the navigation is a breadcrumb trail: > Users / Editors. The main area has a title bar for 'Editors [User group]' with a last modified timestamp (06/10/2002 7:13 pm) and language information (English (United Kingdom)). Below this are buttons for Edit, Move, and Remove. A sidebar on the left lists categories: Users, Guest accounts, Administrator users, Editors (which is selected and highlighted in orange), and Anonymous Users. Under Editors, there are two entries: 'test editor' and 'Administrator User'. At the bottom of the sidebar are buttons for Trash, Small, Medium, and Large. A 'Access control' section contains a link to 'Roles and policies'. The main content area shows 'Assigned roles [2]' with two entries: 'Editor' with 'Subtree (/1/2/)' and 'Editor' with 'Subtree (/1/43/)'. Below this is a table titled 'Available policies [2]':

Role	Module	Function	Limitation
Editor (limited to subtree /1/2/)	content	<i>all functions</i>	No limitations
Editor (limited to subtree /1/2/)	user	login	No limitations
Editor (limited to subtree /1/43/)	content	<i>all functions</i>	No limitations
Editor (limited to subtree /1/43/)	user	login	No limitations

Figure 4.14: The usergroup view interface.

The screenshot above shows a situation when the "Editors" user group is selected. You can bring up a list of roles and policies assigned to this group by enabling the "Roles" and "Policies" windows using the menu at the top.

3. Look at the "Module" column in the table of policies. As long as the "notification" module (page 546) is not listed here, the selected user/group is not allowed to use notifications. Please refer to the next sections for information about how you can create a new role (that grants access to the module) and assign it to a user/group.

Creating a new role

The following text reveals how you can create a new role for granting access to notifications.

1. Click the "User accounts" tab in the administration interface and then access the "Roles and policies" link on the left. You should see the list of existing roles as shown in the screenshot below.

(see figure 4.15)

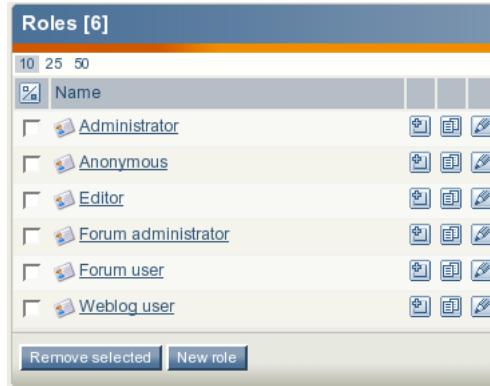


Figure 4.15: The list of roles.

2. Let's create a new role called for example "My notification role". Click the "New role" button under the list of roles. You will be taken to the role edit interface as shown in the following screenshot. (see figure 4.16)

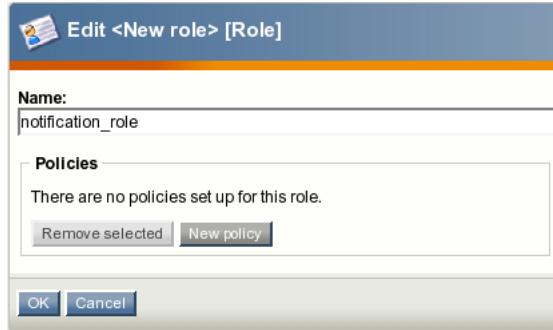


Figure 4.16: Adding a new role.

3. Specify the name of the role and click the "New policy" button.

4. The wizard will help you to create a new policy in two steps. (see figure 4.17)

The above screenshot shows the first step. Select the "notification" module from the drop-down list and click the "Grant access to one function" button.

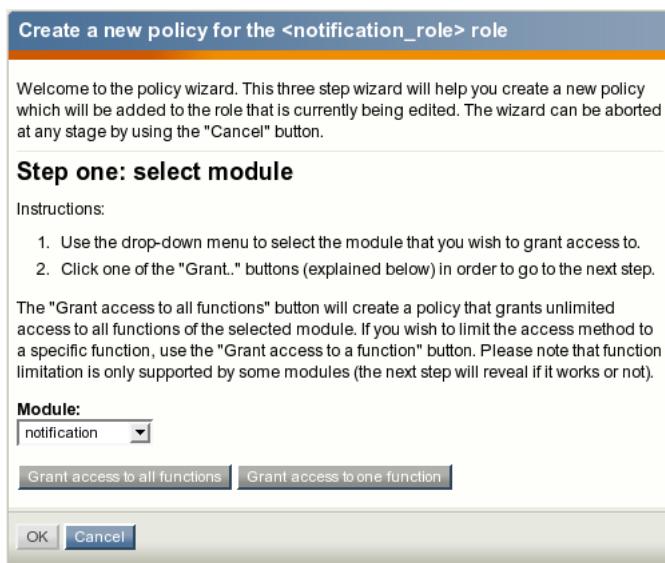


Figure 4.17: The new policy wizard, step 1.

5. You will be taken to the second step as shown in the screenshot below. (*see figure 4.18*)
Choose the "use" function from the dropdown list. Please note that you shouldn't choose the "administrate" function because it grants access to the "runfilter" view of the "notification" module (page 546).
6. Click the "Grant full access" button. (There is no point clicking the "Grant limited access" button because the functions of the "notification" module do not support limitations.)
7. The new policy will appear in the role edit interface as shown in the following screenshot. (*see figure 4.19*)
8. Click "OK" to save your changes and go back the role view interface. (*see figure 4.20*)
The new policy will appear in the role view interface as shown in the screenshot above. You can now assign this role to any user or group (this is explained in the next section).

Assigning a role to a user and/or a user group

A role can be viewed by clicking on its name in the list of existing roles in the role interface (select "Roles and policies" from within the "User accounts" to bring up the role interface).

When you're looking at a role, there should be a list of users/groups towards the bottom of the page. This list reveals the users and groups that the role which is being viewed has been assigned to. The following text explains how to use this list in order to assign the role that is currently being viewed to the "Editors" user group.

1. Click the "Assign" button located under the list of users in the role view interface.

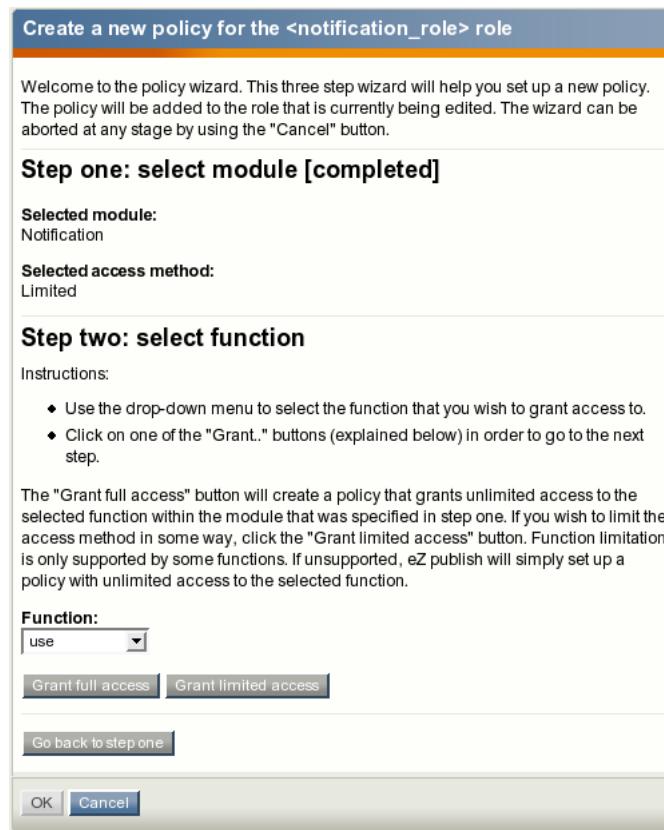


Figure 4.18: The new policy wizard, step 2.



Figure 4.19: The role edit interface.

2. Select the "Editors" user group as shown in the following screenshot and click the "OK" button.
(see figure 4.21)
3. The "Editors" user group will appear in the list of users. The screenshot below shows the

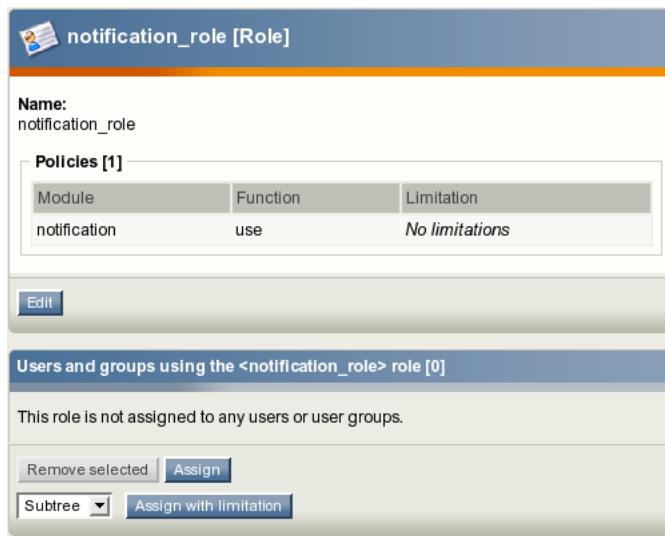


Figure 4.20: The role view interface.



Figure 4.21: Assigning a role to a user group.

role view interface for "My notification role" that is assigned to the "Editors" user group (this means that all users that belong to this group are allowed to use notifications). (see figure 4.22)

Please note that you can assign the role to a single user in the same way as to a user group.

The screenshot shows the configuration interface for a role named "notification_role".

Name: notification_role

Policies [1]

Module	Function	Limitation
notification	use	No limitations

Edit

Users and groups using the <notification_role> role [1]

User/group	Limitation
Editors	No limitations

Buttons: Remove selected, Assign, Subtree, Assign with limitation.

Figure 4.22: The role view interface.

4.1.6 Notification events

The following three notification events are supported by default:

- Publish
- Collaboration
- Current time

Publish

Every time an object is published, a new "ezpublish" event is created.

Collaboration

Every time a collaboration message is generated, a new "ezcollaboration" event is created.

Current time

Every time the "runcronjobs.php" script is executed, a new "ezcurrenttime" event is created. This behavior is specified in the "cronjobs/notification.php" file. The system uses "ezcurrenttime" events for generating digest notifications.

If you need to generate this event manually, add the "notification/runfilter" notation to the URL of your site administration interface and then click the "Spawn time event" button. Please note that the "runfilter" view of the "notification" module (page 546) should be used only for testing and debugging.

The built-in notification event types are stored in the "kernel/classes/notification/event/" directory. It is possible to extend the system by creating custom notification events for special needs.

Creation and storage

Let's say that you have an article on your site, and a user has subscribed for subtree notifications about this article. Every time a new comment is posted or an updated version of the article is published, the system will generate a new "ezpublish" event and store it in the database. This event can be processed by zero, one, or more notification handlers (page 250).

Settings

The available notification event types are specified in the "[NotificationEventTypeSettings]" section of the "notification.ini" (page 1200) configuration file located in the "settings" directory. The following settings can be used under this section:

The "RepositoryDirectories[]" array specifies the directories where eZ Publish will search for built in notification event types. The exact location of the event in the directory is specified using the "AvailableNotificationEventTypes" setting.

The "ExtensionDirectories[]" array specifies the extension directories where eZ Publish will search for additional notification event types. By default eZ Publish will search in the "notificationtypes" subdirectory inside your extension. The exact location of the event in this subdirectory is specified with the "AvailableNotificationEventTypes" setting.

The "AvailableNotificationEventTypes[]" array contains a list of event types.

Example 1

The following lines can be specified in the " notification.ini (page 1200)" configuration file:

```
[NotificationEventTypeSettings]
RepositoryDirectories []=kernel/classes/notification/event/
ExtensionDirectories []
AvailableNotificationEventTypes []=ezpublish
AvailableNotificationEventTypes []=ezcurrenttime
AvailableNotificationEventTypes []=ezcollaboration
```

These settings will make eZ Publish search for the following files for built in notification events:

- kernel/classes/notification/event/ezpublish/ezpublishtype.php
- kernel/classes/notification/event/ezcurrenttime/ezcurrenttimetype.php
- kernel/classes/notification/event/ezcollaboration/ezcollaborationtype.php

Example 2

You can extend the system by creating custom notification events. For example, if you have an extension "nExt" that includes a notification event "nev", put the following lines into an override for " notification.ini (page 1200)" configuration file:

```
[NotificationEventTypeSettings]
ExtensionDirectories []=nExt
AvailableNotificationEventTypes []=nev
```

or

```
[NotificationEventTypeSettings]
RepositoryDirectories []=extension/nExt/notificationtypes/
AvailableNotificationEventTypes []=nev
```

These settings will make eZ Publish expect the additional notification event to be located at "extension/nExt/notificationtypes/nev/nevtype.php"

Please note that you must always clear at least the ini cache in order to make the system re-read the changed configuration files.

4.1.7 Notification handlers

There are several handlers that process notification events. The following handlers are known to the eZ Publish system by default:

- Subtree notification
- General digest
- Collaboration notification

Subtree notification

The "ezsubtree" notification handler processes "ezpublish" events.

General digest

The "ezgeneraldigest" notification handler processes "ezcurrenttime" events.

Collaboration notification

The "ezcollaborationnotification" notification handler processes "ezcollaboration" events.

The built-in notification handlers are stored in the "kernel/classes/notification/handler/" directory. It is possible to extend the system by creating custom notification handlers for special needs.

4

Processing the notification events

Whenever the "eznotificationeventfilter.php" script is executed, the system will try to run every unhandled notification event (page [247](#)) with every available notification handler.

Please note that handling one event may result in sending/generating more than one notification. For example, if a new version of an article is published, all the subscribed users will be notified.

If every notification is sent successfully, the event will be deleted from the database. Otherwise, if some of the generated notifications must be delayed in order to form a daily/weekly/monthly digest, the system will add new notification items to the user collection. A notification item contains data about the notification event, its handler, subscriber e-mail and time when this notification must be sent.

The digest handler starts processing the "ezcurrenttime" event by accessing the collection of notification items. The time specified in the "ezcurrenttime" event will be compared with the time of each notification item in order to determine which items that must be handled at the moment. As long as each notification item contains data on the notification event and its handler, the system will process this event with the right handler. The resulting notifications will be collected into digest messages and sent to the subscribers.

If the notification item was handled successfully, this item will be removed from the collection. If none of the remaining notification items reference the handled notification event, this event will be deleted from the database.

The "ezcurrenttime" event will be deleted from the database when processing is completed. Please note that processing the "ezcurrenttime" event by the digest handler does not always result in sending/generating digest notifications (for example, if none of the subscribers has chosen the digest mode for notifications).

Settings

The "[NotificationEventHandlerSettings]" section of the "notification.ini (page 1200)" configuration file defines the event handlers that will respond to the notification event. Under this section, the following settings can be specified:

The "RepositoryDirectories[]" array specifies the directories where eZ Publish will search for built in notification handlers. The exact location of the handler in the directory is specified using the "AvailableNotificationEventTypes" setting.

The "ExtensionDirectories[]" array specifies the extension directories where eZ Publish will search for additional notification handlers. By default eZ Publish will search in the "notification/handler/" subdirectory inside your extension. The exact location of the handler in this subdirectory is specified using the "AvailableNotificationEventTypes" setting.

The "AvailableNotificationEventTypes[]" array contains a list of handlers.

Example 1

The following lines can be specified in the "notification.ini (page 1200)" configuration file:

```
[NotificationEventHandlerSettings]
RepositoryDirectories []=kernel/classes/notification/handler/
ExtensionDirectories []
AvailableNotificationEventTypes []=ezgeneraldigest
AvailableNotificationEventTypes []=ezcollaborationnotification
AvailableNotificationEventTypes []=ezsubtree
```

These settings will make eZ Publish search for the following files for built in notification handlers.

- kernel/classes/notification/handler/ezgeneraldigest/ezgeneraldigesthandler.php
- kernel/classes/notification/handler/ezcollaborationnotification/ezcollaborationnotificationhandler.php
- kernel/classes/notification/handler/ezsubtree/ezsubtreehandler.php

Example 2

You can extend the system by creating custom notification handlers. For example, if you have an extension "nExt" that includes a notification handler "nh" put the following lines into an override for the "notification.ini (page 1200)" configuration file:

```
[NotificationEventHandlerSettings]
ExtensionDirectories [] =nExt
AvailableNotificationEventTypes [] =nh
```

or

```
[NotificationEventHandlerSettings]
RepositoryDirectories [] =extension/nExt/notification/handler/
AvailableNotificationEventTypes [] =nh
```

These settings will make eZ Publish expect the additional notification handler to be located at "extension/nExt/notification/handler/nh/nh.handler.php"

4.1.8 Frequently Asked Questions

Q: Is there a standard user who automatically get notified about all the site changes (creation/modification of content objects)?

A: By default, none of the users is notified about all the site changes. If you want to be notified whenever content is changed or added, you can subscribe for subtree notifications for the top node in the "Content structure" tree.

Q: Is it possible to be notified about new user registrations?

A: You can subscribe for subtree notifications for the top node in the "User accounts" tree so that you will be notified every time a new user is created. To do this, click on the "User accounts" tab in the administration interface, locate the desired node (under which new users are created upon user registration) in the tree and select "Add to my notifications" using the context menu.

Q: Is it possible to receive E-mails whenever I need to approve an article? Is it possible that the writer of the article is notified whether or not the article was approved?

A: It is possible to get notifications when you need to approve an item (same for the author). This can be easily done by enabling the collaboration notifications. Currently there is no support for notifications to the author when the article has been approved/rejected.

Q: I have subscribed for notifications but I do not receive any E-mails.

A: You might have forgotten about "runcronjobs.php" script. If you wish to use the notification system, this script must be executed periodically.

Q: I use both subtree and collaboration notifications. The subtree notifications work well but I do not receive collaboration notifications.

A: The collaboration notifications are sent every time a new collaboration message is generated. Check your collaboration messages by clicking the "My Account" tab in the administration interface and select the "Collaboration" link on the left. If there are no collaboration messages there, check your collaboration settings by clicking the "Setup" tab and choosing the "Workflows" and/or "Triggers" link on the left. Please refer to the "Workflows (page 163)" and "Approve (page 773)" documentation chapters for more information about workflows, triggers and approval events. (A simple example of implementing an approval mechanism including [creating a workflow, connecting it to a trigger function](#) and [approving entries](#) can be found in the old documentation.)

Q: Notification settings are not available for one of my users.

A: A user must be logged in to access the notification settings. If the notification settings are still unavailable after logging in, please check the role/policy settings specified for the user(s) as described in the "Granting access to notifications (page 241)" part of the documentation.

Q: Why do the users see the "access denied" page when they click the "Keep me updated" button?

A: Perhaps they are not allowed to use notifications. Check the role/policy settings specified for these users as described in the "Granting access to notifications (page 241)" chapter.

Q: I have the default/built-in forum on my site and I wish that every registered user should be able to subscribe/unsubscribe for subtree notifications about the forum/topic/reply. How can I do this?

A: By default, all the users that belong to the "Guest accounts" user group are allowed to use notifications. This is specified in the default "Forum user" role that is assigned to the guest user group. It is possible to assign this role to other users (please refer to "Assigning a role to a user / user group" section of the "Granting access to notifications (page 241)" chapter for more information). There is no point to assign this role to the "Administrator users" group. The default "Administrator" role assigned to the "Administrator users" group allows these users to access all modules including the "notification" module (page 546).

Q: In the role/policy settings I can choose the "administrate" function when granting access to the "notification" module. Does it mean that it is possible to view/edit the notification settings of each subscribed user somewhere in the administration area?

A: Although letting administrators to view and/or edit notification settings for all users is probably good idea, it is not implemented yet. The only difference between "use" and "administrate" functions is that the latter grants access to the "runfilter" view of the "notification" module (page 546). Please note that this view should only be used for testing and debugging.

Q: Is it possible to force digest mode for notifications so that the digest mode is set by default for all the subscribed users (with the preset time)?

A: This functionality is not implemented. By default, the digest mode is off and the database contains no records about this setting. If the user sets the digest mode, it will be recorded in the database.

Q: Is there any way to set "filters" for subtree notifications? I have a set of articles under a certain folder and the users are notified whenever a new article is created there. However, they also receive notifications when an existing article is edited or a new folder is created. I'd like to specify "only notify if a new object of type article is being created" or something similar.

A: This is not supported at the moment.

4.2 Search engine

The system comes with a built-in search engine which integrates tightly with the content structure. It is capable of indexing everything that is inputted through the native content model.

In eZ Publish, a content class describes the actual data structures (for example news articles, products, etc.). The classes are built up of attributes which are represented by datatypes. An attribute can be the title of an article, the price of a product and so on. It is possible to control which attributes that should be indexed by the search engine. This can be done by making use of the "Searchable" checkboxes while editing a class. Some datatypes (for example float, price, etc.) do not support indexing. Please refer to the datatype overview (page 268) page to see which datatypes that can be indexed.

When an object is published, the attributes that are marked searchable will be indexed by the search engine. It will then be possible to use the search interface to find words or phrases that are a part of the published content. For example, if the user searches for "backpack", the system will return a list of all kinds of objects where the word "backpack" occurs. This is the default behavior. The following screenshot shows the standard search interface.

(see figure 4.23)

The screenshot shows the standard search interface. At the top, there is a search bar with the text "ez" and a "Search" button. Below the search bar, a message says "For more options try the [Advanced search](#)". The main area is titled "Search for <ez> returned 6 matches". It contains a table with two columns: "Name" and "Type". The results are:

Name	Type
eZ publish	Folder
Products	Folder
Forums	Folder
Galleries	Folder
Media files	Folder
Flash demo	Flash

Figure 4.23: Standard search interface

Advanced search

The advanced search interface makes it possible tweak and narrow the search. The following features are supported:

- Search for several words at the same time (for example "car bike train").
- Search for an exact phrase (for example "cheap cars in Scandinavia").
- Class level filtering (limit the search to a specific class).

- Attribute level filtering (search only a specific attribute).
- Tree level filtering (limit the search to a part of the node tree).
- Section filtering (limit the search to objects that belong to a certain section).
- Time filtering (yesterday, last week/month/3-months/year).

The following screenshot shows the advanced search interface.

(see figure 4.24)

The screenshot shows the 'Advanced search' interface. It includes fields for searching words or phrases, dropdowns for 'Class' (set to 'Any class') and 'In' (set to 'Any section'), and dropdowns for 'Class attribute' and 'Published' (both set to 'Update attributes'). A 'Search' button is at the bottom.

Figure 4.24: Advanced search interface

Wildcard searching

The default behavior of the search engine is that it only searches for complete words or phrases. If the user searches for "demo", the system will not return objects that contain words like "demolition", "demonstration" and so on. However, eZ Publish does in fact support wildcard searching, but it must be turned on by adding the following lines to a configuration override for "site.ini":

```
[SearchSettings]  
EnableWildcard=true
```

When the wildcard search feature is turned on, it is possible to use the asterisk character as a wildcard, for example like this: "demo*". In this case, eZ Publish will return a list of objects that contain words starting with "demo". For example, it would return objects containing words like "demonstration", "demolition", etc. When this notation is used, the result will also return objects that contain the word which was specified before the asterisk. In other words, objects containing only the word "demo" will also be returned.

Please note that the asterisk can only be used after a word. This means that the following search queries are invalid: "*demo" and "some*thing".

Warning! There is a good reason for the wildcard search being turned off by default. It requires a lot more processing time than the standard search. This means that the server might have to be upgraded in order to produce faster results and to achieve less overall system load.

Logical operators

Inline logical operators like "AND" and "OR" are not supported. This means that it is not possible to specify search queries like "cars AND minivans" or "trucks OR vans". However, it is in fact possible to do an AND search. This can be done by making use of the "Search for all of the following words" input field in the advanced search interface. For example, if the user inputs "cars bikes" then the system will return a list of objects that contain both of these words. The order of the words is insignificant.

Search statistics

The setup part of the administration interface provides access to a page that reveals information about words/phrases that have been searched along with the average results that have been returned. The following screenshot shows the search statistics interface.

(see figure 4.25)

Search statistics		
Phrase	Number of phrases	Average result returned
ez publish	2	6.00
some code	1	1.00
some	1	10.00
ez	1	6.00
ez publish news	1	0.00
ez news	1	0.00
news article	1	0.00
news	1	2.00
news demo	1	2.00

Figure 4.25: *Search statistics*

The "Reset statistics" button will simply clear the search log.

4.3 WebDAV

WebDAV is an abbreviation for "Web-based Distributed Authoring and Versioning" (published as an open standard under RFC 2518). WebDAV is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on a web server. This is achieved by making use of a WebDAV compatible client / application. For example, it is possible to use recent versions of KDE's Konqueror or Microsoft's Internet Explorer. Using a WebDAV compatible client, the user connects to the server and is able to browse and manage files in a similar way as with a network share or an FTP server. In other words, what this protocol does is that it makes it possible to browse, create, remove, upload, download, rename, etc. files and directories on a web server. One of the most important advantages of this technology is that it uses port 80 for network traffic. This means that if you are able to surf the site from your workstation, you can also use WebDAV to administer it. It does not require firewall reconfiguration.

eZ Publish and WebDAV

From version 3.2 and up, eZ Publish provides a built in WebDAV server. This implementation allows users to communicate with eZ Publish using a WebDAV compatible client. Once connected, it is possible to browse and manage the node tree of a site. The tree will be displayed as if it were a filesystem (as directories and files).

When a user connects to the eZ Publish WebDAV server for the first time, the system will display a list of the siteaccesses that have been made available for WebDAV. This list can be configured using the "SiteList[]" directive under "[SiteSettings]" in a configuration override for "site.ini". Please note that the system does not ask for a username/password combination at this stage. In other words, anyone with network access will be able to see the names of the available siteaccesses. The following screenshot shows what the user will see if there are two available siteaccesses, "example" and "plain_user".

(see figure 4.26)

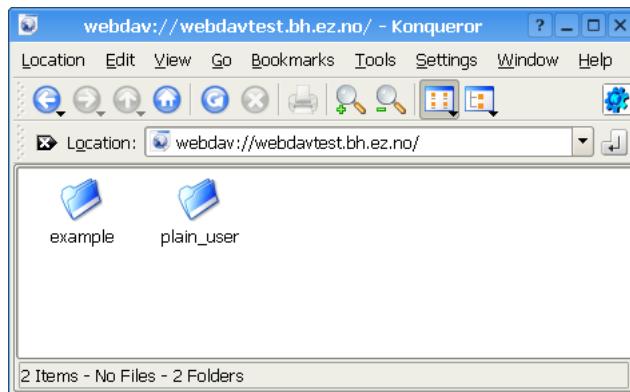


Figure 4.26: WebDAV - Virtual top folder

When a siteaccess is chosen, the system will attempt to authenticate the user by asking for a

username/password combination. The next screenshot shows this.

(see figure 4.27)



Figure 4.27: WebDAV - Login

The provided username and password must belong to a valid eZ Publish user that exists for the selected siteaccess. Furthermore, the user must have sufficient privileges in order to be able to see the contents of the node tree. The following screenshot shows a WebDAV client displaying the contents of the root node of an eZ Publish siteaccess called "plain_user".

(see figure 4.28)

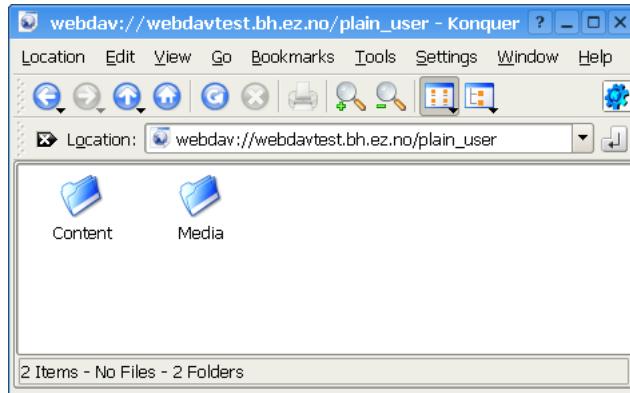


Figure 4.28: WebDAV - Top level nodes

As the screenshot indicates, the user will be able to browse and manage the contents of the "Content" and "Media" top level nodes. The next screenshot shows an example of what the user will see after doing some exploration of the node structure of the "Content" top level node.

(see figure 4.29)

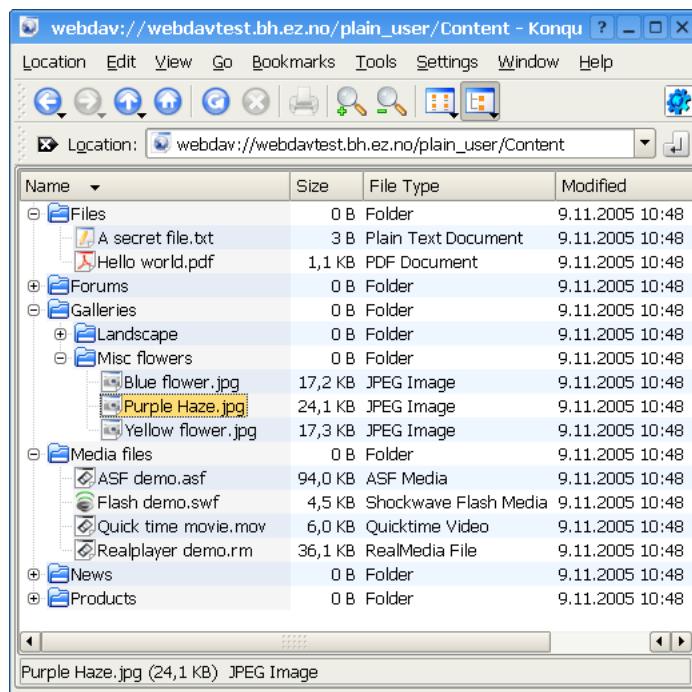


Figure 4.29: WebDAV - Content node tree

Browsing and downloading

The default behavior is that all nodes are displayed as directories. The reason for this is because in eZ Publish any object can be placed under any other object by the way of nodes. Displaying the nodes as directories makes it possible to explore the structure of the node tree. However, not all nodes are displayed as directories.

Nodes that reference objects containing datatypes that store files will be displayed as files instead of directories. This means that for example nodes that make use of the image, media or the file datatype will be displayed as files. When downloaded, eZ Publish will send the file that is contained in the attribute which is represented by a datatype capable of storing a file. If several attributes are represented by such datatypes, it is the contents of the first attribute that will be used.

The "FolderClasses[]" directive in "webdav.ini" can be used to configure which types of nodes that should be shown as directories in the WebDAV client. The default configuration assures that nodes referencing folder objects are always displayed as directories. Adding a class that makes use of a datatype capable of storing a file will result in an override of the behavior described in the previous paragraph. In other words, this setting makes it possible to display different types of nodes as directories even if they contain files.

Uploading

Any type of file can be uploaded to the system. Files will be stored using instances of the file class. In other words, every time a file is uploaded, eZ Publish will create a file object where the file attribute will contain the uploaded data. In addition, a node will be created at the location where the file was uploaded in the tree. This is the default behavior. However, not all file types will be created as file objects.

It is possible to configure the system so that it creates different kinds of objects based on the type of the uploaded file. For example, the default configuration makes sure that uploaded images are created as image objects. This behavior is controlled by mapping MIME types to classes. The mappings can be configured using the "MimeTypeMap[]" directive under "CreateSettings" in a configuration override for "upload.ini". The "DefaultClass" directive determines which class that should be used if there is no suitable mapping. This is usually set to "file", which means that unrecognized file types will be created as file objects. The following example shows the default mappings.

```
MimeTypeMap []
MimeTypeMap[image]=image
MimeTypeMap[video/quicktime]=quicktime
MimeTypeMap[video/x-msvideo]=windows_media
MimeTypeMap[video/vnd.rn-realvideo]=real_video
MimeTypeMap[application/vnd.rn-realmedia]=real_video
MimeTypeMap[application/x-shockwave-flash]=flash
```

Each entry in the "MimeTypeMap[]" array must be further specified using a block that reveals details about the class that is being mapped to. The blocks must contain the following information:

- The identifier of the target class (appended with "_ClassSettings").
- The class attribute identifier which will get the file data.
- The class attribute identifier which will get the name.
- A pattern that defines the name of the object.

The following example shows the default class map block for images.

```
[image_ClassSettings]
FileAttribute=image
NameAttribute=name
NamePattern=<original_filename_base>
```

The example above will tell eZ Publish that when an image is uploaded, the actual file data should be put into an attribute identified by the string "image". The name of the image should be stored using an attribute called "name" (as before, it is the identifier of the attribute that is

used). The "NamePattern" tells the system about how it should generate the name for uploaded images. It may contain plain text and special tags enclosed by angle brackets ("<" and ">"). The following table reveals the tags that can be used.

Tag	Description
original_filename	The name of the uploaded file, like it was on the local machine (for example "test.jpg").
original_filename_base	The name of the uploaded file without an extension (for example "test").
original_filename_suffix	The extension of the uploaded file (for example ".jpg").
mime_type	The MIME type of the uploaded file (for example "image/jpeg").

Custom upload handling

It is possible to use custom upload handlers in order to process uploaded files in a special way. Custom upload handlers must be provided as extensions. A handler must be automatically triggered whenever a certain type of file is uploaded to the system. This can be done by making use of the "MimeTypeHandlerMap[]" directive under "[CreateSettings]" in "upload.ini". For example, the following line will make sure that uploaded images (regardless of type) are handled by a class called "ezimageuploadhandler" located in "ezimageuploadhandler.php".

```
MimeTypeHandlerMap[image]=ezimageuploadhandler
```

It is also possible to have only a specific type of file be processed by the upload handler. The following example demonstrates how to only handle JPEG images.

```
MimeTypeHandlerMap[image/jpeg]=ezimageuploadhandler
```

The upload handler itself must be put in a directory called "uploadhandlers" in an extension, like this:

```
eZ Publish  
|  
-extensions  
|  
-example  
|  
-uploadhandlers  
|  
-ezimageuploadhandler.php
```

The following code shows the skeleton of a custom upload handler.

```
include_once( 'kernel/classes/ezcontentuploadhandler.php' );

class eZExampleUploadHandler extends eZContentUploadHandler
{
    function eZExampleUploadHandler()
    {
        $this->eZContentUploadHandler( 'Example file handling', 'example' );
    }

    /**
     * Handles the uploading of example files.
     */
    function handleFile( &$upload, &$result,
                        $filePath, $originalFilename, $mimeInfo,
                        $location, $existingNode )
    {
        // Implement your import/conversion routine here
        copy( $filepath, "var/cache/example.jpeg" );
    }
}
```

4.3.1 Setting it up

This section describes how eZ Publish can be configured in order to function as a WebDAV server. Please note that the DNS and the web server also needs to be configured.

Step 1: Enable the WebDAV server

The master WebDAV switch must be turned on. Create a global configuration override for "webdav.ini" and make sure that it contains the following lines:

```
[GeneralSettings]
EnableWebDAV=true
```

Step 2: Add the desired siteaccesses

In order to allow WebDAV access for a specific siteaccess, the name of the siteaccess must be specified in the "SiteList[]" array under "[SiteSettings]" in a configuration override for "site.ini". Make sure that the global configuration override for "site.ini" contains the necessary lines. The following example shows how WebDAV can be opened up for a siteaccess called "plain_user" and another one called "example".

```
[SiteSettings]
SiteList[]
SiteList[] = plain_user
SiteList[] = example
```

Step 3: Clear all caches

The eZ Publish part of the configuration is done. Clear all caches in order to make sure that the system uses the updated version of the configuration.

Step 4: Setup a DNS entry

Set up a DNS entry (for example a subdomain) that will be used to access the WebDAV server. The entry must point to the IP address of the web server. For example, if you're using "www.example.com" to access the web pages, you could set up "webdav.example.com" for WebDAV.

Step 5: Configure the web server

There is a file called "webdav.php" in the root of the eZ Publish directory. This file provides the actual WebDAV interface. The web server must automatically execute this file whenever a

WebDAV client sends a command to the server. The following lines show an example of how this can be done in the configuration file of the Apache web server.

```
<Virtualhost 128.39.140.28>
    <Directory /path/to/ezpublish>
        Options FollowSymLinks Indexes ExecCGI
        AllowOverride None
    </Directory>
    DocumentRoot /path/to/ezpublish
    RewriteEngine On
    RewriteRule . /webdav.php
    ServerAdmin admin@example.com
    ServerName webdav.example.com
</VirtualHost>
```

Note: make sure that you have a "NamedVirtualHost" line before the declaratoin of the virtual hosts.

Step 6: Test

Launch a WebDAV compatible client / application and attempt to connect to the server.

Internet Explorer

Recent versions of Microsoft's Internet Explorer (6.0.2800.1106 or later) contain a built-in WebDAV client. The target address must be opened as a web folder.

1. Start Internet Explorer.
2. Access the "File" menu and select "Open", a dialog should appear.
3. Type in the address of the WebDAV server along with a hash ("#") character at the end, like this: `http://webdav.example.com/#`
(see figure 4.30)

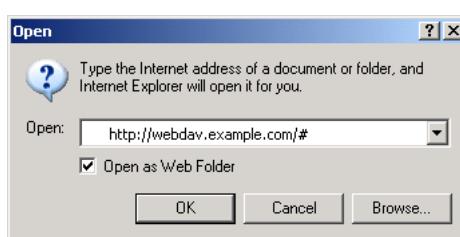


Figure 4.30: WebDAV - IE open dialog

4. Make sure that the "Open as web folder" checkbox is checked.
5. Click OK. You should be able to see the available siteaccesses as directories.

KDE/Konqueror

Make sure you have a recent version of Konqueror (3.1.3 or later). Open up a Konqueror window and attempt to browse the WebDAV server by accessing it using a URL that resembles the following example: "webdav://webdav.example.com/".

(see figure 4.31)

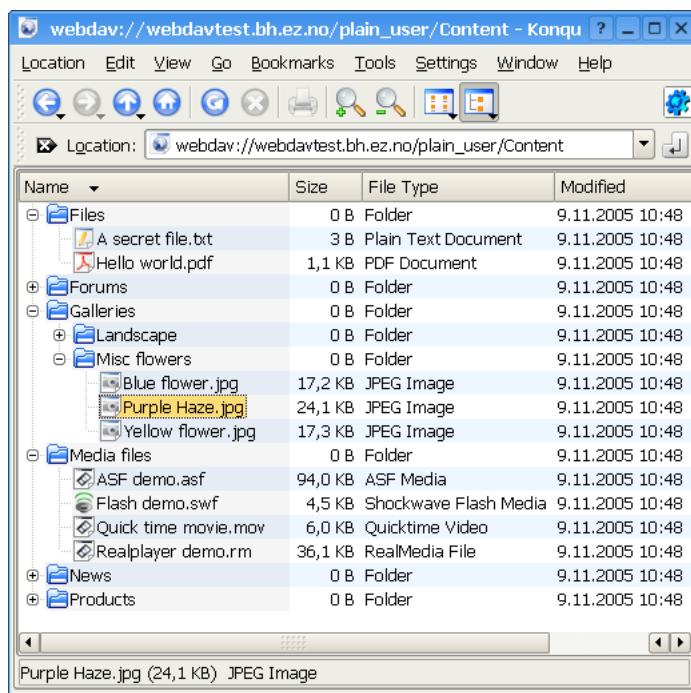


Figure 4.31: WebDAV - Content node tree

Chapter 5

Reference

This chapter provides reference information for developers. It covers the following topics:

- Datatypes (page [268](#))
- Content classes (page [344](#))
- Modules (page [371](#))
- Views (page [695](#))
- Objects (page [696](#))
- Workflow events (page [772](#))
- Template operators (page [780](#))
- Template functions (page [998](#))
- Template control structures (page [1050](#))
- Template override conditions (page [1065](#))
- Template fetch functions (page [1088](#))
- Template PDF functions (page [1089](#))
- Configuration files (page [1132](#))
- Libraries (page [1449](#))
- XML tags (page [1462](#))

5.1 Datatypes

Authors (page 270)

Stores info about additional authors.

Checkbox (page 272)

Stores a binary value (on or off).

Date (page 274)

Validates and stores a date value.

Date and time (page 276)

Validates and stores a date and a time value.

E-mail (page 278)

Validates and stores an E-mail address.

Enum (page 279)

DEPRECATED

File (page 280)

Stores any type of file.

Float (page 284)

Validates and stores a decimal value.

Identifier (page 286)

Generates a non-editable identification string.

Image (page 288)

Validates and stores a digital image.

Ini setting (page 292)

DEPRECATED

Integer (page 293)

Validates and stores an integer value.

ISBN (page 295)

Validates and stores an ISBN value.

Keywords (page 297)

Stores keywords.

Matrix (page 299)

Stores multiple rows and columns of text.

Media (page 301)

Stores a media file (Flash/QT/Real/etc.).

Multi-option (page 304)

Allows option selections. [Webshop]

Object relation (page 306)

Stores a relation to a content object.

Object relations (page 308)

Stores relations to other content objects.

Option (page 310)

Allows an option selection. [Webshop]

Price (page 312)

Stores a price (inc/ex VAT). [Webshop]

Range option (page 314)

Allows an integer selection. [Webshop]

Selection (page 316)

Stores single and multiple choices.

Subtree subscription (page 318)

DEPRECATED

Text block (page 319)

Stores multiple lines of unformatted text.

Text line (page 321)

Stores a single line of unformatted text.

Time (page 323)

Validates and stores a time value.

URL (page 325)

Validates and stores a URL / address.

User account (page 327)

Validates and stores info about a user.

XML block (page 329)

Validates and stores multiple lines of formatted text.

5.1.1 Authors

Summary

Stores info about additional authors.

Properties

Name	Internal name	Searchable	Information collector
Authors	ezauthor	Yes.	No.

Description

This datatype allows the validation, storage and retrieval of additional authors. For each author, it is capable of handling a name and an E-mail address. It is only the E-mail address that will be validated. It is typically useful when there is a need for storing information about additional authors who have written/created different parts of an object's contents. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.1)

The screenshot shows a form for editing a class attribute. At the top, it says "9. Additional authors [Author] (id:249)". Below that, there are two input fields: "Name:" containing "Additional authors" and "Identifier:" containing "additional_authors". At the bottom, there are several checkboxes: "Required" (checked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked).

Figure 5.1: Class attribute edit interface for the "Authors" datatype.

As the screenshot indicates, the "Authors" datatype does not have any class specific configuration parameters.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.2)

When a new object is created, the attribute using this datatype will have its first row set to the name and the E-mail address of the user who created the object.

Additional authors	
Name	E-mail
<input type="checkbox"/> Administrator User	allman@example.com
<input type="checkbox"/> Art Vandalay	art@example.com

Figure 5.2: Object attribute edit interface for the "Authors" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezauthor (page 699) object.

5.1.2 Checkbox

Summary

Stores a binary value (on or off).

Properties

Name	Internal name	Searchable	Information collector
Checkbox	ezboolean	Yes.	Yes.

Description

This datatype allows the storage and retrieval of a binary value. It can be either on/true or off/false. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.3)

The screenshot shows a configuration dialog for a class attribute named "Show comments". The "Name" field contains "Show comments" and the "Identifier" field contains "show_comments". Under the "Required" and "Searchable" checkboxes, there is a checked mark. The "Information collector" and "Disable translation" checkboxes are unchecked. A "Default value" section contains a checked checkbox.

Figure 5.3: Class attribute edit interface for the "Checkbox" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.4)



Figure 5.4: Object attribute edit interface for the "Checkbox" datatype.

Default value

The "Default value" parameter makes it possible to control the initial value of an attribute using this datatype when a new object is created.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns either "1" or "0" depending on the state of the checkbox (checked or unchecked).

5.1.3 Date

Summary

Validates and stores a date value.

Properties

Name	Internal name	Searchable	Information collector
Date	ezdate	Yes.	No.

Description

This datatype allows the validation, storage and retrieval of dates consisting of a year, month and day value. The valid input range is 01.01.1970 - 19.01.2038. The following screenshot shows the class attribute edit interface of this datatype.

(see figure 5.5)

The screenshot shows a software interface for editing a class attribute named "Date of birth". At the top, it says "5. Date of birth [Date field] (id:258)". Below that are fields for "Name" (containing "Date of birth") and "Identifier" (containing "date_of_birth"). There are several checkboxes: "Required" (checked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked). At the bottom, there is a "Default value" dropdown menu set to "Empty".

Figure 5.5: Class attribute edit interface for the "Date" datatype.

Default value

The "Default value" parameter has two options: "Empty" and "Current date". The default setting is "Empty", which means that when a new object is created, the attribute using this datatype will be empty. If the "Current date" setting is used, the current date will be set when a new object is created.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.6)

A screenshot of a web-based form interface for editing an object attribute. The specific field shown is for a date of birth. The label "Date of birth" is at the top. Below it, there are three input fields: "Year:" containing "1971", "Month:" containing "12", and "Day:" containing "22". All three fields have a light blue background and a white border.

Figure 5.6: Object attribute edit interface for the "Date" datatype.

Raw output

The ".content" of an ezcontentobjectattribute object using this datatype returns an ezdate (page 728) object.

5.1.4 Date and time

Summary

Validates and stores a date and a time value.

Properties

Name	Internal name	Searchable	Information collector
Date and time	ezdatetime	Yes.	No.

Description

This datatype allows the validation, storage and retrieval of a date/time value. It is capable of storing a date/time consisting of a year, month, day, hour and minute value. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.7)

Figure 5.7: Class attribute edit interface for the "Datetime" datatype.

Default value

The "Default value" parameter can be used to control the initial value of an attribute using this datatype when a new object is created. There are three options:

- Empty
- Current datetime
- Adjusted current datetime

The default setting is "Empty", which means that when a new object is created, the attribute using this datatype will be empty. If the "Current datetime" option is used, the current date and time will be set. If the "Adjusted datetime" is used, an adjusted value of the current date and time will be set. How much the current date and time should be adjusted must be specified using the fields within the "Current datetime adjusted by" group.

Current datetime adjusted by

The fields within this group can be used to specify the desired date/time adjustment when the ("Adjusted current datetime") option is used. Both positive and negative numerical values are allowed. If the values given in the example above are used and an object is created at 00:00 on the first of January 2005, the initial value of the attribute will be set to 04:05, third of February, 2006.

Object attribute edit interface

The following screenshot shows the object attribute interface for this datatype.

(see figure 5.8)

A screenshot of a web-based form interface. At the top, it says "Time of arrival". Below that is a horizontal row of five input fields: "Year:", "Month:", "Day:", "Hour:", and "Minute:". Each field has a small number inside it: "Year: 2000", "Month: 05", "Day: 02", "Hour: 10", and "Minute: 00".

Figure 5.8: Object attribute edit interface for the "Date and time" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezdatetime (page 729) object.

5.1.5 E-mail

Summary

Validates and stores an E-mail address.

Properties

Name	Internal name	Searchable	Information collector
E-mail	ezemail	Yes.	Yes.

Description

This datatype allows the validation, storage and retrieval of an electronic mail address. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.9)

A screenshot of a web-based form for editing a class attribute. The title bar says "7. E-mail address [Email] (id:260)". The form has two main sections: "Name:" containing "E-mail address" and "Identifier:" containing "email". Below these are several checkboxes: "Required" (checked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked).

Figure 5.9: Class attribute edit interface for the "Email" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.10)

A screenshot of a web-based form for editing an object attribute. It contains a single text input field labeled "E-mail address:" with the value "info@ez.no".

Figure 5.10: Object attribute edit interface for the "E-mail" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object returns the actual E-mail address.

5.1.6 Enum

Summary

DEPRECATED

Properties

Name	Internal name	Searchable	Information collector
Enum	ezenum	Yes.	No.

Description

This datatype should not be used any more because it is slow. It has been substituted by the "Selection" (page [316](#)) datatype.

5.1.7 File

Summary

Stores any type of file.

Properties

Name	Internal name	Searchable	Information collector
File	ezbinaryfile	Yes.	No.

Description

This datatype allows the storage and retrieval of a single file. It is capable of handling virtually any file type and is typically used for storing legacy document types such as PDF files, Word documents, spreadsheets, etc. The maximum allowed file size is determined by the "Max file size" class attribute edit parameter and the "upload_max_filesize" directive in the main PHP configuration file ("php.ini"). The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.11)

The screenshot shows a software interface for editing a class attribute. At the top, it says "10. Sonar stream [Binary file] (id:250)". Below that, there are several input fields and checkboxes:

- Name:** Sonar stream
- Identifier:** sonar_stream
- Required:**
- Searchable:**
- Information collector:**
- Disable transla:**
- Max file size:** 16 MB

Figure 5.11: Class attribute edit interface for the "File" datatype.

Max file size

The "Max file size" parameter makes it possible to set the highest size (in megabytes) that the system will allow. By default, this parameter is zero, which means that eZ publish will not do any size checking when files are uploaded. In the example above, the "Max file size" parameter is set to 16 MB, which means that the object edit interface will not allow the upload of files that are larger than 16 megabytes. However, if the value of the "upload_max_filesize" PHP setting is lower than 16 megabytes, the underlying system will cancel the upload.

Object attribute edit interface

The following screenshot shows the object attribute edit interface when an attribute using this datatype does not contain any file.

(see figure 5.12)

The screenshot shows a form titled "Sonar stream (required)". It has a section for "Current file:" which displays the message "There is no file." Below this is a "Remove" button. A "New file for upload:" input field is present with a "Choose" button next to it.

Figure 5.12: Object attribute edit interface for the "File" datatype.

The following screenshot shows the object attribute edit interface when an attribute using this datatype contains a file. The interface reveals the name of the file that was uploaded ("BD-Scratch.wav"), the MIME type ("audio/wav") and the size (1.15 MB).

(see figure 5.13)

The screenshot shows a form titled "Sonar stream (required)". It has a table for "Current file:" with three columns: "Filename", "MIME type", and "Size". The table row shows "BDScratch.wav", "audio/wav", and "1.15 MB". Below the table is a "Remove" button. A "New file for upload:" input field is present with a "Choose" button next to it.

Figure 5.13: Object attribute edit interface for the "File" datatype.

MIME types

The MIME type will be automatically set based on the extension of the uploaded file's name. If the extension is unknown, the default MIME type will be used ("application/octet-stream"). The MIME types can be configured at the end of the "/lib/ezutils/classes/ezmimetype.php" file.

Storage

The uploaded files are stored on the filesystem. The main reason for this is because the filesystem is much faster than the database when it comes to the storage and retrieval of large data chunks. Having the files on the filesystem allows the webserver to serve them directly without the need of going through the database. In addition, this technique makes it easier to use external tools to

manipulate/scan/index the contents of the uploaded files and it dramatically decreases the size of the database.

All files uploaded through an attribute that makes use of the file datatype will be stored below "storage/original" within the directory specified by the "VarDir" directive in a configuration override for "site.ini". A new subdirectory will be created for every MIME type. For example, if an executable (.exe) file is uploaded, a directory called "application" will be created; if a text file is uploaded then a directory called "text" will be created, and so on. The uploaded files will be put in the different MIME type directories. Instead of re-using the original filenames, eZ publish will create a hash for every file. The following illustration shows the location of two uploaded files (an .exe and a .txt file) when the var directory is set to "my_site".

(see figure 5.14)



Figure 5.14: Complete directory structure with uploaded files.

The system keeps track of the files using a database table called "ezbinaryfile" consisting of the following fields:

Field	Description
content_object_attribute_id	The identification of the content object attribute.
download_count	The number of times the file has been downloaded.
filename	The name of the file on the filesystem (for example "5fd39fbaf751369965a4108715d5dea9.txt").
mime_type	The MIME type of the file (for example "text/plain").
original_filename	The original name of the uploaded file (for example "readme.txt").
version	The version of the object that the file belongs to.

Binary file indexing

eZ publish is capable of indexing the actual contents of uploaded files. This feature makes it possible to use the built-in search engine to search for something that is inside a file; for example the contents of a PDF file or a spreadsheet. By default, the system is only capable of indexing the

contents of plain text/ASCII files. However, by making use of external programs, it is capable of indexing the contents of virtually any file type (as long as there is a program that goes through the file and returns keywords/contents as plain text). The external handlers can be set up in a configuration override for the "binaryfile.ini" file.

Raw output

The ".content" of an ezcontentobjectattribute (page [717](#)) object using this datatype returns an ezbinaryfile (page [702](#)) object.

5.1.8 Float

Summary

Validates and stores a decimal value.

Properties

Name	Internal name	Searchable	Information collector
Float	ezfloat	No.	No.

Description

This datatype makes it possible to validate, store and retrieve a single decimal value. It is capable of handling both positive and negative numbers ranging from -3.402823466E+38 to 3.402823466E+38. Please note that these numbers may vary depending on the platform and the database that is used. The following screenshot shows the class edit interface for this datatype.

(see figure 5.15)

The screenshot shows a software interface for editing a class named "Temperature [Float] (id:252)". At the top, there are buttons for moving up and down in a list. Below that, there are several input fields and checkboxes:

- Name:** Temperature
- Identifier:** temperature
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Default value:** 24.13
- Min float value:** -40.00
- Max float value:** 120.00

Figure 5.15: Class edit interface for the "Float" datatype.

Default value

The "Default value" parameter makes it possible to set a default decimal value. When the parameter is used and a new object is created, the contents of the attribute using this datatype will be preset to the given value. In the example above, the "Temperature" attribute of new objects will be set to 24.13.

Min integer value

The "Min float value" parameter makes it possible to set the lowest value that the input interface will allow. The default value of this parameter is empty, which means that the system will allow the lowest possible value (-3.402823466E+38). In the example above, the parameter is set to "-40.00". This means that the input interface will not allow the storage of numbers with values less than -40.00.

Max float value

The "Max float value" parameter makes it possible to set the highest value that the input interface will allow. The default value of this parameter is empty, which means that the system will allow the highest possible value (3.402823466E+38). In the example above, the parameter is set to "120.00". This means that the input interface will not allow the storage of numbers with values as high as 120.00.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.16)

A screenshot of a software interface showing a single-line text input field. The label "Temperature:" is positioned above the input field. Inside the input field, the value "18.50" is displayed.

Figure 5.16: Object attribute edit interface for the "Float" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual decimal value.

5.1.9 Identifier

Summary

Generates a non-editable identification string.

Properties

Name	Internal name	Searchable	Information collector
Identifier	ezidentifier	Yes.	No.

Description

This datatype allows the automatic generation of unique identification strings for objects. An instance of a class that makes use of this datatype will have a unique identification string generated whenever the object is published for the very first time. The identification string is system-wide and may consist of the following elements:

- User configurable start-text
- Automatically generated identification number (configurable start value and number of digits)
- User configurable end-text

The system will increment the actual counter(s) whenever a new object is published. It will not decrement and reorganize the identification strings when an object is removed. The identification strings are generated and maintained by the system and thus they can not be modified using the object edit interface. The identification strings are generated based on the class type and the identification number of the class attribute. In other words, the identifiers do not depend on the objects' locations within the content node tree. In addition, the datatype may be used to represent several attributes within the same class. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.17)

Pretext

The "Pretext" parameter can be used to specify a desired start-text (all characters are allowed) that should appear before the automatically generated identification number. In the example above, the text "ABC" is used. This means that the identification strings generated for instances of this class will start with the letters "ABC".

The screenshot shows a software interface for editing a class attribute named 'Product ID'. At the top, there's a header bar with a back arrow, the title '11. Product ID [Identifier] (id:264)', and navigation icons for download and upload. Below the header, the 'Name:' field contains 'Product ID' and the 'Identifier:' field contains 'product_id'. There are several checkboxes: 'Required' (unchecked), 'Searchable' (checked), 'Information collector' (unchecked), and 'Disable translation' (unchecked). At the bottom, there are four input fields labeled 'Pretext:', 'Posttext:', 'Digits:', and 'Start value'. The 'Pretext:' field has 'ABC' in it, 'Posttext:' has 'XYZ', 'Digits:' has '2', and 'Start value' has '1'.

Figure 5.17: Class attribute edit interface for the "Identifier" datatype.

Posttext

The "Posttext" parameter can be used to specify a desired end-text (all characters are allowed) that should appear after the automatically generated identification number. In the example above, the text "XYZ" is used. This means that the identification strings generated for instances of this class will end with the letters "XYZ".

Digits

The "Digits" parameter makes it possible to insert additional zeros in order to generate equally long identification strings. In the example above, the digits parameter is set to "2". This means that numbers below 10 will appear with a prepended zero: "01", "02", "03"..."09". If the parameter was set to "3", the system would generate "001", "002" and so on. The "Digits" parameter can not be used to set the actual range / stop value. It only makes it possible to prepend the actual number with zeros.

Start value

The "Start value" parameter can be used to specify a desired start value for the counter; the default value is zero. In the example above, the value "1" is used. This means that the counter will start at "1". Given all the parameters above, the identification string generated for the very first object will be "ABC01XYZ".

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual identification value.

5.1.10 Image

Summary

Validates and stores a digital image.

Properties

Name	Internal name	Searchable	Information collector
Image	ezimage	No.	No.

Description

This datatype allows the storage of digital images. It is capable of handling virtually any image type. The maximum allowed file size is determined by the "Max file size" class attribute edit parameter and the "upload_max_filesize" directive in the main PHP configuration file ("php.ini"). The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.18)

The screenshot shows a web-based form for editing a class attribute. At the top, there's a header bar with a back arrow, the title '1. Image [Image] (id:204)', and download/upload icons. Below the header are several input fields and checkboxes:

- Name:** A text input field containing 'Image'.
- Identifier:** A text input field containing 'Image'.
- Required:** A checked checkbox.
- Searchable:** An unchecked checkbox.
- Information collector:** An unchecked checkbox.
- Disable translation:** An unchecked checkbox.
- Max file size:** A text input field containing '4' followed by a dropdown menu showing 'MB'.

Figure 5.18: Class attribute edit interface for the "Image" datatype.

Max file size

The "Max file size" parameter makes it possible to set the highest size (in megabytes) that the system will allow. By default, this parameter is zero, which means that eZ Publish will not do any size checking when image files are uploaded. In the example above, the "Max file size" parameter is set to 16 MB, which means that the object edit interface will not allow the upload of images that are larger than 16 megabytes. However, if the value of the "upload_max_filesize" PHP setting is lower than 16 megabytes, the underlying system will cancel the upload.

Object attribute edit interface

The following screenshot shows the object attribute edit interface when an attribute using this datatype does not contain an image.

(see figure 5.19)

The screenshot shows a form titled 'Image'. It has a section for 'Current image' which says 'There is no image file.' and includes a 'Remove image' button. Below this is a 'New image file for upload:' input field with a 'Choose' button. At the bottom is an 'Alternative image text:' input field.

Figure 5.19: Object attribute edit interface for the "Image" datatype.

The following screenshot shows the object attribute edit interface when an attribute using this datatype contains an actual image. The interface reveals the image itself, the name of the file that was uploaded ("dj_cat.jpg"), the MIME type ("image/jpeg") and the size (29.02 kB).

(see figure 5.20)

The screenshot shows a form titled 'Image'. It has a section for 'Current image' showing a thumbnail of a cat named 'dj_cat.jpg' with details: Filename: dj_cat.jpg, MIME type: image/jpeg, Size: 29.02 kB. Below this is a 'Remove image' button. At the bottom are 'New image file for upload:' and 'Alternative image text:' input fields, with the latter containing the text 'The D.J. Cat Show'.

Figure 5.20: Object attribute edit interface for the "Image" datatype.

Storage

The uploaded images are stored on the filesystem. The main reason for this is because the filesystem is much faster than the database when it comes to the storage and retrieval of large data chunks. Having the images on the filesystem allows the webserver to serve them directly

without the need of going through the database. In addition, this technique makes it easier to use external tools to manipulate the images and it dramatically decreases the size of the database.

All images uploaded through an attribute that makes use of the image datatype will be stored below "storage/images" within the directory specified by the "VarDir" directive in a configuration override for "site.ini". A directory structure is generated for each object that stores an image. The structure will be an exact copy of the actual node path (consisting of the object names) from the root to the main node that references the object which contains images. The following illustration shows the path to a directory on the filesystem in which the different images of an object are stored.

(see figure 5.21)

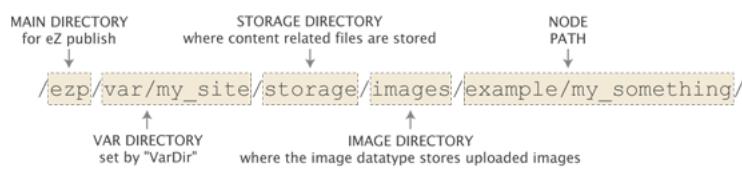


Figure 5.21: Example of image path on the filesystem.

In the example above, an image has been stored by an object called "My Something". The main node of this object is a child of a node called "Example". The var directory used by the siteaccess is "var/my_site". The images stored by the "My Something" object will be located in this directory. For each image, version and translation, a new directory will be created. The following illustration shows the naming convention of the directories under which the actual images are stored.

(see figure 5.22)

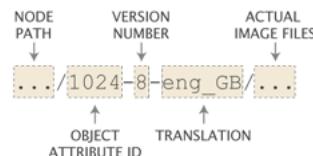


Figure 5.22: Example of an image subdirectory.

The example above shows a directory for an image that has been stored by the 1024th object attribute. The image belongs to the British translation for the eight version of the content object. The following illustration shows a complete directory structure with actual image files.

(see figure 5.23)

As the illustration indicates, the image stored by the 1024th attribute (that belongs to the "My Something" object) is represented by multiple files. The original image is "test.png", the rest of the images are the different image variations.



Figure 5.23: Complete directory structure with uploaded image and generated variations.

Image variations

With the help of an external application (ImageMagick) or the PHP image library (GD), eZ Publish is capable of generating different variations of an uploaded image. This feature is typically useful when there is a need to show the same image in different ways (for example different sizes). The image variations are controlled by different configuration directives in image.ini. The following table shows the default image variations.

Variation	Width	Height
reference	max 600 pixels	max 600 pixels
small	max 100 pixels	max 100 pixels
medium	max 200 pixels	max 200 pixels
large	max 300 pixels	max 300 pixels
rss	max 88 pixels	max 31 pixels

When an image variation is created (and the default settings are used), eZ Publish will generate a reference image using the file that was uploaded. The reference image is then used to generate the different image variations (small, medium, large and rss) on-demand using the constraints specified in the table above. The generated variations will be cached on the filesystem.

Depending on the image system used, eZ Publish is capable of generating image variations that consist of much more than just scaling. For example, it is possible to generate sharpened grayscale images for the thumbnails of a photo album. Please refer to the documentation of the image.ini configuration file for more information about the different possibilities.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezimagealiashandler (page 730) object.

5.1.11 Ini setting

Summary

DEPRECATED

Properties

Name	Internal name	Searchable	Information collector
Ini setting	ezinisetting	Yes.	No.

Description

This datatype should no longer be used.

5.1.12 Integer

Summary

Validates and stores an integer value.

Properties

Name	Internal name	Searchable	Information collector
Integer	ezinteger	Yes.	No.

Description

This datatype makes it possible to validate, store and retrieve a single integer value. It is capable of handling both positive and negative numbers ranging from -2,147,483,648 to 2,147,483,647. The following screenshot shows the class edit interface for this datatype.

(see figure 5.24)

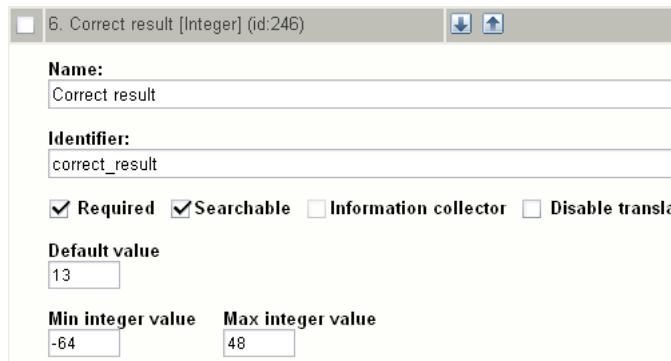


Figure 5.24: Class edit interface for the "Integer" datatype.

Default value

The "Default value" parameter makes it possible to set a default integer. When the parameter is used and a new object is created, the contents of the attribute using this datatype will be preset to the given value. In the example above, the "Correct result" attribute of new objects will be set to 13.

Min integer value

The "Min integer value" parameter makes it possible to set the lowest value that the input interface will allow. The default value of this parameter is empty, which means that the system will

allow the lowest possible value, which is -2,147,483,648. In the example above, the parameter is set to "-64". This means that the input interface will not allow the storage of numbers with values less than -64.

Max integer value

The "Max integer value" parameter makes it possible to set the highest value that the input interface will allow. The default value of this parameter is empty, which means that the system will allow the highest possible value, which is 2,147,483,647. In the example above, the parameter is set to "48". This means that the input interface will not allow the storage of numbers with values as high as 48.

Object attribute edit interface

The following screenshot shows the object attribute interface for this datatype.

(see figure 5.25)

Correct result:	<input type="text" value="13"/>
-----------------	---------------------------------

Figure 5.25: Object attribute edit interface for the "Integer" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual integer.

5.1.13 ISBN

Summary

Validates and stores an ISBN value.

Properties

Name	Internal name	Searchable	Information collector
ISBN	ezisbn	Yes.	No.

Description

The "ISBN" datatype allows the validation, storage and retrieval of a single ISBN (International Standard Book Number) value. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.26)

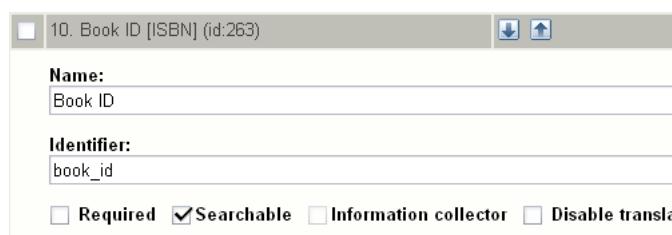


Figure 5.26: Class attribute edit interface for the "ISBN" datatype.

Object attribute edit interface

The following screenshot shows the object attribute interface for this datatype.

(see figure 5.27)

Book ID:
ISBN - - -

Figure 5.27: Object attribute interface for the "ISBN" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page [717](#)) object returns a string containing the actual ISBN value.

5.1.14 Keywords

Summary

Stores keywords.

Properties

Name	Internal name	Searchable	Information collector
Keywords	keywords	Yes.	No.

Description

This datatype allows the storage of keywords for an object. The keywords must be specified as a comma separated list of words and/or phrases. This datatype can be used to connect objects of the same type based on their keywords. It is typically useful when it comes to creating interfaces that allow the user to quickly browse other pages with related content (for example "see also" or "related pages" list of hyperlinks). If two objects share at least one common keyword, the objects will be connected behind the scenes. The system will create the necessary entries in the "ezkeyword" and the "ezkeyword_attribute_link" database tables. Please note that this datatype does not generate object relations, it simply uses its own tables in the database to connect the objects. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.28)

The screenshot shows a software interface for editing a class attribute. At the top, it says "13. Keywords [Keyword] (id:266)". Below that is a "Name:" field containing "Keywords" and an "Identifier:" field containing "keywords". Underneath these fields are four checkboxes: "Required" (unchecked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked).

Figure 5.28: Class attribute edit interface for the "Keywords" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.29)

Keywords:

penguins, scooters, megaphones

Figure 5.29: Object attribute edit interface for the "Keywords" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezkeyword (page 738) object.

5.1.15 Matrix

Summary

Stores multiple rows and columns of text.

Properties

Name	Internal name	Searchable	Information collector
Matrix	ezmatrix	Yes.	No.

Description

This datatype allows the storage and retrieval of information structured in a table. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.30)

The screenshot shows the 'Class attribute edit interface' for the 'Matrix' datatype. At the top, there's a title bar with the identifier '14. Electronic components [Matrix] (id:267)' and navigation icons. Below the title are several input fields and checkboxes:

- Name:** Electronic components
- Identifier:** electronic_components
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**

Below these are two more input fields:

- Default name:** Transistor
- Default number of rows:** 1

At the bottom, there's a section titled 'Columns' containing three entries:

Matrix column	Identifier
Name	name
Part number	part_number
Location	location

Buttons for 'Remove selected' and 'New column' are located at the bottom of this section.

Figure 5.30: Class attribute edit interface for the "Matrix" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.31)

Electronic components			
Name	Part number	Location	
CPU	1	Skien	
RAM	2	Odessa	
HD	3	Dortmund	
<input type="button" value="Remove selected"/>		5	<input type="button" value="Add rows"/>

Figure 5.31: *Object attribute edit interface for the "Matrix" datatype.*

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezmatrix (page 742) object.

5.1.16 Media

Summary

Stores a media file (Flash/QT/Real/etc.).

Properties

Name	Internal name	Searchable	Information collector
Media	ezmedia	No.	No.

Description

This datatype allows the storage and playback of a video file. It is capable of handling Apple QuickTime, Macromedia Flash, Microsoft Windows Media and Real Media files. The maximum allowed file size is determined by the "Max file size" class attribute edit parameter and the "upload_max_filesize" directive in the main PHP configuration file ("php.ini"). The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.32)

A screenshot of a web-based form for editing a class attribute. The title bar says "2. Movie clip [Media] (id:203)". The form has several input fields and checkboxes:

- Name:** Movie clip
- Identifier:** movie_clip
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Media player type:** QuickTime (dropdown menu)
- Max file size:** 32 MB

Figure 5.32: Class attribute edit interface for the "Media" datatype.

Media player type

The "Media player type" parameter controls the way the video file will be played back on the client side. The following options are available:

- Flash
- QuickTime
- Real player
- Windows media player

Max file size

The "Max file size" parameter makes it possible to set the highest size (in megabytes) that the system will allow. By default, this parameter is zero, which means that eZ Publish will not do any size checking when files are uploaded. In the example above, the "Max file size" parameter is set to 32 MB, which means that the object edit interface will not allow the upload of files that are larger than 32 megabytes. However, if the value of the "upload_max_filesize" PHP setting is lower than 32 megabytes, the underlying system will cancel the upload.

Object attribute edit interface

The object attribute edit interface for the "Media" datatype depends on the selected "Media player type" (in class edit mode). The following screenshots show the different object attribute edit interfaces for the supported media types.

(see figure 5.33)

The screenshot shows the 'Movie clip' configuration for a Flash file. The 'Type:' field is set to 'Flash'. Under 'Current file:', there is a table:

Filename	MIME type	Size
ezpublish-mediademo.swf	application/x-shockwave-flash	2.74 kB

Below the table is a 'Remove' button. A 'New file for upload:' input field is present with a 'Choose' button. At the bottom, there are settings for 'Width:' (640), 'Height:' (480), 'Quality:' (High), 'Autoplay:' (checked), and 'Loop:' (checked).

Figure 5.33: Object attribute edit interface for the "Media" datatype (Flash).

(see figure 5.34)

The screenshot shows the 'Movie clip' configuration for a QuickTime file. The 'Type:' field is set to 'QuickTime'. Under 'Current file:', there is a table:

Filename	MIME type	Size
ezpublish-mediademo.mov	video/quicktime	364.00 kB

Below the table is a 'Remove' button. A 'New file for upload:' input field is present with a 'Choose' button. At the bottom, there are settings for 'Width:' (640), 'Height:' (480), 'Controller:' (unchecked), 'Autoplay:' (checked), and 'Loop:' (checked).

Figure 5.34: Object attribute edit interface for the "Media" datatype (QuickTime).

(see figure 5.35)

Movie clip

Type:
Real player

Current file:

Filename	MIME type	Size
ezpublish-mediademo.ram	application/vnd.rn-realmedia	364.00 kB

New file for upload:

Width: Height: Controls: Autoplay:

Figure 5.35: Object attribute edit interface for the "Media" datatype (Real Media).

(see figure 5.36)

Movie clip

Type:
Windows media player

Current file:

Filename	MIME type	Size
ezpublish-mediademo.avi	video/x-msvideo	364.00 kB

New file for upload:

Width: Height: Controller: Autoplay:

Figure 5.36: Object attribute edit interface for the "Media" datatype (Windows media).

Storage

Files that have been uploaded through the "Media" datatype are stored in the same way as files that are uploaded using the "File" (page 280) datatype. However, the system keeps track of the media files using the "ezmedia" table instead of the "ezbinaryfile" table.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezmedia (page 745) object.

5.1.17 Multi-option

Summary

Allows option selections. [Webshop]

Properties

Name	Internal name	Searchable	Information collector
Multi-option	ezmultioption	Yes.	No.

Description

This datatype makes it possible to create multiple groups of options for each content object. Each option can be assigned a short text and an additional price. This datatype works in the same way as the "Option" (page 310) datatype. The only difference is that instead of supporting only one group of options, it allows the creation of multiple groups of options for each content object. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.37)

The screenshot shows a software interface for editing a class attribute. At the top, a header bar displays the title "16. Product options [Multi-option] (id:269)" and navigation icons for download and upload. Below the header, there are two input fields: "Name:" containing "Product options" and "Identifier:" containing "product_options". Underneath these fields are several checkboxes: "Required" (unchecked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked). At the bottom right of the form, there are small "down" and "up" arrow buttons for rearranging items.

Figure 5.37: Class attribute edit interface for the "Multi-option" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.38)

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezmultioption (page 747) object.

Product options

Option set name:
Details

<input type="checkbox"/> 0	Priority: 1
----------------------------	-------------

Name:
Color

Options:

Option	Additional price	Default
<input type="checkbox"/> Red	10	<input checked="" type="radio"/>
<input type="checkbox"/> Green	20	<input type="radio"/>

Remove selected **Add option**

<input type="checkbox"/> 1	Priority: 2
----------------------------	-------------

Name:
Size

Options:

Option	Additional price	Default
<input type="checkbox"/> Small	10	<input type="radio"/>
<input type="checkbox"/> Medium	20	<input checked="" type="radio"/>
<input type="checkbox"/> Large	30	<input type="radio"/>

Remove selected **Add option**

Remove selected **Add multioption**

Figure 5.38: Object attribute edit interface for the "Multi-option" datatype.

5.1.18 Object relation

Summary

Stores a relation to a content object.

Properties

Name	Internal name	Searchable	Information collector
Object relation	ezobjectrelation	Yes.	No.

Description

This datatype allows the relation of a single object. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.39)

The screenshot shows the '17. Related object [Object relation] (id:270)' class attribute edit interface. It includes fields for Name (Related object), Identifier (related_object), and checkboxes for Required, Searchable, Information collector, and Disable translation. The Selection method is set to 'Browse'. A table for Default selection item shows one entry: AAA - Folder (Type: Folder, Section: Standard). There are buttons for Remove selection and Select item. An Allow fuzzy match checkbox is also present.

Figure 5.39: Class attribute edit interface for the "Object relation" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.40)

Related object		
Name	Type	Section
Article	Article	Standard
Remove object		Find object

Figure 5.40: *Object attribute edit interface for the "Object relation" datatype.*

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns either FALSE (if there is no relation) or an ezcontentobject (page 712) object.

5.1.19 Object relations

Summary

Stores relations to other content objects.

Properties

Name	Internal name	Searchable	Information collector
Object relations	ezobjectrelationlist	Yes.	No.

Description

This datatype allows the relation of multiple objects. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.41)

The screenshot shows a software interface for editing a class attribute. At the top, it says "10. Object list [Object relations] (id:195)". Below that are several input fields and checkboxes:

- Name:** Object list
- Identifier:** object_list
- Allowed classes:** A dropdown menu containing "Any", "Folder", "Article", and "User group".

Below the identifier field are four checkboxes:

- Required
- Searchable
- Information collector
- Disable translation

Figure 5.41: Class attribute edit interface for the "Object relations" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.42)

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns either FALSE (if there are no relations) or an array of ezcontentobjectattribute (page 712) objects.

Object list			
Name	Type	Section	Order
Article	Article Standard	1	<input type="text"/>
Another article	Article Standard	2	<input type="text"/>

Remove selected **Add existing objects**

Figure 5.42: *Object attribute edit interface for the "Object relations" datatype.*

5.1.20 Option

Summary

Allows an option selection. [Webshop]

Properties

Name	Internal name	Searchable	Information collector
Option	ezoption	No.	Yes.

Description

This datatype makes it possible to create a single group of options for each content object. Each option can be assigned a short text and an additional price. For example, it can be used to sell T-shirts in different colors where the price is different for some (or all) colors. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.43)

The screenshot shows a software interface for editing a class attribute named 'Product option [Option] (id:272)'. The interface has a header with a back button and two upward arrows. It contains several input fields and checkboxes:

- Name:** Product option
- Identifier:** product_option
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Default name:** Color

Figure 5.43: Class attribute edit interface for the "Option" datatype.

The "Default name" parameter can be used to specify a name which will be used (as the name for the option) every time a new object is created.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.44)

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezoption (page 750) object.

Product options (required)

Name:	Color											
Options:	<table border="1"><thead><tr><th>Option</th><th>Additional price</th></tr></thead><tbody><tr><td><input type="checkbox"/> Black</td><td>0</td></tr><tr><td><input type="checkbox"/> Red</td><td>10</td></tr><tr><td><input type="checkbox"/> Green</td><td>20</td></tr><tr><td><input type="checkbox"/> Blue</td><td>30</td></tr></tbody></table>		Option	Additional price	<input type="checkbox"/> Black	0	<input type="checkbox"/> Red	10	<input type="checkbox"/> Green	20	<input type="checkbox"/> Blue	30
Option	Additional price											
<input type="checkbox"/> Black	0											
<input type="checkbox"/> Red	10											
<input type="checkbox"/> Green	20											
<input type="checkbox"/> Blue	30											
<input type="button" value="Remove selected"/> <input type="button" value="Add option"/>												

Figure 5.44: Object attribute edit interface for the "Option" datatype.

5.1.21 Price

Summary

Stores a price (inc/ex VAT). [Webshop]

Properties

Name	Internal name	Searchable	Information collector
Price	ezprice	No.	No.

Description

This datatype allows the storage of a price and thus makes it possible to connect content objects with the e-commerce subsystem. The e-commerce features of eZ Publish are described in the "Webshop" (page 159) section of the "Concepts and basics" chapter. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.45)

Figure 5.45: Class attribute edit interface for the "Price" datatype.

Parameters and usage

This is a special datatype which plugs more deeply into the system. Instances of any class containing the price datatype will automatically be treated as products. A class attribute represented by the price datatype makes use of one of the predefined VATs. There are two ways in which the selected VAT can be used. This configuration depends on how the product prices are entered when the objects are created. The first alternative (Price inc. VAT) is to be used if the prices that are entered already include the value added tax. The second alternative (Price ex. VAT) should be used if the prices that are entered do not contain the value added tax. When the first alternative is used and the product is viewed, the price that was entered will be shown.

When the second alternative is used and the product is viewed, the price will be the price that was entered plus the VAT. When the object is in the basket and the basket is viewed, it is possible to see the price of the products with and without the VATs (regardless of which approach that was used). The VAT parameters on the class level only control the default VAT settings that will be used when a new object is created. In other words, the VAT settings may be modified for each individual product / object.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.
(see figure 5.46)

The screenshot shows a form titled "Price (required)". It contains three input fields: "Price:" with the value "129.50", "VAT:" with the dropdown menu open showing "Price inc. VAT", and "VAT type:" with the dropdown menu open showing "Std, 22%".

Figure 5.46: Object attribute edit interface for the "Price" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezprice (page 756) object.

5.1.22 Range option

Summary

Allows an integer selection. [Webshop]

Properties

Name	Internal name	Searchable	Information collector
Range option	ezrangeoption	No.	No.

Description

This datatype makes it possible to create a single group of enumerated options for each content object. For example, it can be used in a scenario where the goal is to sell shoes of different sizes and the size does not affect the price. For each content object, the administrator needs to set up the available range (if any). The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.47)

The screenshot shows the 'Range option' class attribute edit interface. At the top, there is a header bar with a back arrow, the title '3. Range option [Range option] (id:186)', and up/down navigation icons. Below the header, there are several input fields:

- Name:** Airborne Ranger
- Identifier:** airborne_ranger
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Default name:** Shoe size

Figure 5.47: Class attribute edit interface for the "Range option" datatype.

The "Default name" parameter can be used to specify a name which will be used (as the name for the option) every time a new object is created.

5

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.48)

Airborne Ranger (required)

Name:
Shoe size

Start value: Stop value: Step value:

32 48 1

Figure 5.48: Object attribute edit interface for the "Range option" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezrangeoption (page 758) object.

5.1.23 Selection

Summary

Stores single and multiple choices.

Properties

Name	Internal name	Searchable	Information collector
Selection	ezselection	Yes.	No.

Description

This datatype allows the storage of single or multiple option selections. The options must be defined on the class level. This datatype can for example be used to define different categories for news articles (as shown in the example below: "Local", "National", etc.). The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.49)

The screenshot shows the 'Category [Selection]' class attribute edit interface. At the top, there are buttons for moving items up and down. Below that, there are fields for 'Name' (containing 'Category') and 'Identifier' (containing 'category'). There are also checkboxes for 'Required' (unchecked), 'Searchable' (checked), 'Information collector' (unchecked), and 'Disable translation' (unchecked). A 'Style' dropdown is set to 'Single choice'. In the 'Options' section, 'Option' is selected. Below it, four options are listed: 'Local', 'National', 'International', and 'Interplanetary'. At the bottom of the 'Options' section are buttons for 'Remove selected' and 'New option'.

Figure 5.49: Class attribute edit interface for the "Selection" datatype.

Style

The "Style" parameter controls the behavior of the object attribute edit interface. It can be either "Single" or "Multiple". While "Single" means that the object attribute edit interface will only

allow a single selection, the "Multiple" setting allows the selection of multiple options.

Options

The "Options" interface allows the specification of the options that should be available for selection when an object is edited. The options are identified by an identification number. The identification number of the first option is zero. Please note that the system does not give any warning if an option that is used is removed. In other words, removing options may result in an inconsistent database.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.50)



Figure 5.50: Object attribute interface for the "Selection" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an array of the identification numbers (as strings) of the selected options.

5.1.24 Subtree subscription

Summary

DEPRECATED

Properties

Name	Internal name	Searchable	Information collector
Subtree subscription	ezsubtreesubscription	No.	No.

Description

This datatype should no longer be used. It has been replaced by the "My notification settings" interface of the "My account" part in the administration interface.

5.1.25 Text block

Summary

Stores multiple lines of unformatted text.

Properties

Name	Internal name	Searchable	Information collector
Text block	eztext	Yes.	Yes.

Description

This datatype allows the storage and retrieval of multiple lines of unformatted text. It is capable of handling up to 16,777,216 characters. The following screenshot shows the class edit interface for this datatype.

(see figure 5.51)

The screenshot shows a software interface for editing a class named "Message". At the top, it says "4. Message [Text field] (id:244)". Below that are two input fields: "Name:" containing "Message" and "Identifier:" containing "message". Underneath these are several checkboxes: "Required" (checked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked). At the bottom, there is a "Preferred number of rows:" dropdown set to "10".

Figure 5.51: Class edit interface for the "Text block" datatype.

Preferred number of rows

The "Preferred number of rows" parameter makes it possible to control the height of the input field that is displayed when an object is being edited. The following options are available: 2, 5, 10, 15, 20 and 25.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.52)

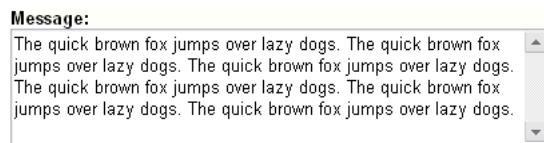


Figure 5.52: Object attribute edit interface for the "Text block" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual text.

5.1.26 Text line

Summary

Stores a single line of unformatted text.

Properties

Name	Internal name	Searchable	Information collector
Text line	ezstring	Yes.	Yes.

Description

This datatype makes it possible to store and retrieve a single line of unformatted text. It is capable of handling up to 255 number of characters. The following screenshot shows the class edit interface for this datatype.

(see figure 5.53)

The screenshot shows a software interface for editing a class named "3. Summary [Text line] (id:198)". The interface includes the following fields:

- Name:** Summary text
- Identifier:** summary_text
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Default value:** The summary has not yet been added.
- Max string length:** 64 characters

Figure 5.53: Class edit interface for the "Text line" datatype.

Default value

The "Default value" parameter makes it possible to set a default text. When the parameter is used and a new object is created, the contents of the attribute using this datatype will be preset to the given text. In the example above, the "Summary" attribute of new objects will be set to "The summary has not yet been added."

Max string length

The "Max string length" parameter makes it possible to control the maximum number of characters that the input interface should allow. By default, this parameter is empty - which means

that the system will allow the maximum number of characters (255). In the example above, the parameter is set to 64. This means that the input interface will not allow the storage of strings that exceed 64 characters.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.54)



Figure 5.54: Object attribute interface for the "Text line" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual text.

5.1.27 Time

Summary

Validates and stores a time value.

Properties

Name	Internal name	Searchable	Information collector
Time	eztime	No.	No.

Description

This datatype allows the validation, storage and retrieval of a time value consisting of hour and minute. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.55)

The screenshot shows a software interface for editing a class attribute named "Departure". At the top, it says "27. Departure [Time field] (id:280)". Below that are several input fields and checkboxes:

- Name:** Departure
- Identifier:** departure
- Required:**
- Searchable:**
- Information collector:**
- Disable translation:**
- Default value:** Empty

Figure 5.55: Class attribute edit interface for the "Time" datatype.

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.56)

The screenshot shows a software interface for editing an object attribute named "Departure". It features a large text input field labeled "Departure" containing the text "11:58". Below the input field, there are two smaller text boxes labeled "Hour:" and "Minute:", each containing the value "11" and "58" respectively.

Figure 5.56: Object attribute edit interface for the "Time" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page [717](#)) object using this datatype returns an eztime (page [763](#)) object.

5.1.28 URL

Summary

Validates and stores a URL / address.

Properties

Name	Internal name	Searchable	Information collector
URL	ezurl	No.	No.

Description

This datatype allows the validation and storage of a hyperlink. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.57)

The screenshot shows a web-based form for editing a class attribute. At the top, it says "23. Hyperlink [URL] (id:276)". Below that are two input fields: "Name:" containing "Hyperlink" and "Identifier:" containing "hyperlink". At the bottom of the form are four checkboxes: "Required", "Searchable", "Information collector", and "Disable translation". Above the checkboxes are two small buttons: a downward arrow and an upward arrow.

Figure 5.57: Class attribute edit interface for the "URL" datatype.

Object attribute edit interface

For each link, an address and an additional text may be stored. The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.58)

The screenshot shows a web-based form for editing an object attribute. It has a single input field labeled "URL" containing the value "http://ez.no". Below it is another input field labeled "Text" containing the value "eZ systems - The creators of eZ publish".

Figure 5.58: Object attribute edit interface for the "URL" datatype.

URL storage and checking

URLs that are input either using the URL or the XML block datatype are stored and handled in a special way. This solution makes it possible to view, edit and check if the URLs actually work on a massive scale. Please refer to the "URL storage" (page 138) section of the "Concepts and basics" (page 96) chapter for more information about this feature.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns a string containing the actual address.

5.1.29 User account

Summary

Validates and stores info about a user.

Properties

Name	Internal name	Searchable	Information collector
User account	ezuser	Yes.	No.

Description

This datatype allows storage and retrieval of a single user account. Whenever an object that uses this datatype is published for the first time, the system will create a new user. The user's ID number will be the identification number of the object. The following screenshot shows the class attribute edit interface for this datatype.

(see figure 5.59)

A screenshot of a software interface titled "24. User [User account] (id:277)". It shows two input fields: "Name:" containing "User" and "Identifier:" containing "user". Below the fields are four checkboxes: "Required" (checked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked).

Figure 5.59: Class attribute edit interface for the "User account" datatype.

Object attribute edit interface

For each user, the "User account" datatype stores the following information:

- A username
- A password
- An E-mail address

The username can only be modified before the object is published for the first time. The following screenshot shows the object attribute edit interface for this datatype.

(see figure 5.60)

User account (required)

User ID:
196

Username: Password: Confirm password: E-mail:

Figure 5.60: Object attribute edit interface for the "User account" datatype.

User account settings

The properties of a user account can be tweaked by the administrator. This can be done by following the "Change user account settings" link from within the view of a user account object in the administration interface. Currently it is only possible to change whether the user account should be enabled or disabled (whether the user is allowed to log in to the system or not). The following screenshot shows the user account settings panel.

(see figure 5.61)

User settings for <Anonymous User>

Maximum concurrent logins:
1000

Enable user account:

OK | Cancel

Figure 5.61: Settings interface for the "User account" datatype.

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezuser (page 765) object.

5.1.30 XML block

Summary

Validates and stores multiple lines of formatted text.

Properties

Name	Internal name	Searchable	Information collector
XML block	ezxmltext	Yes.	No.

Description

Although there are no immediate visual clues, this datatype behaves quite differently compared to the regular "Text block" datatype. In particular, it is capable of validating and storing multiple lines of formatted text instead of just plain text. The text in an XML block must be formatted using a collection of predefined tags. The tags control the actual HTML markup of the content. eZ Publish comes with a collection of tags that cover the needs of typical everyday tasks. In addition, it is also possible to extend the system by creating custom tags for special needs. By default, the datatype supports the following:

- Headings (page [331](#))
- Bold text (page [332](#))
- Italic text (page [333](#))
- Unformatted text (page [334](#))
- Lists (page [335](#))
- Tables (page [336](#))
- Hyperlinks (page [337](#))
- Anchors (page [339](#))
- Object embedding (page [340](#))
- Custom tags (page [342](#))
- Paragraphs (page [343](#))

The following screenshot shows the class attribute edit interface for this datatype.

(see figure [5.62](#))

The screenshot shows the configuration interface for a class attribute named "Description". It includes fields for "Name" (set to "Description") and "Identifier" (set to "description"). There are checkboxes for "Required" (unchecked), "Searchable" (checked), "Information collector" (unchecked), and "Disable translation" (unchecked). A dropdown menu for "Preferred number of rows" is set to 10.

Figure 5.62: *Class attribute edit interface for the "XML block" datatype.*

Object attribute edit interface

The following screenshot shows the object attribute edit interface for this datatype.
(see figure 5.63)

The screenshot shows a rich text editor for an object attribute. The text area contains the story "The quick brown fox jumps over lazy dogs" with specific parts highlighted by XML tags: "**fox**" and "[dogs](eznode://1313)".

Figure 5.63: *Object attribute edit interface for the "XML block" datatype.*

Raw output

The ".content" of an ezcontentobjectattribute (page 717) object using this datatype returns an ezxmltext (page 771) object.

Headings

Headings/titles can be added by making use of either the "h" or the "header" tag. The "level" parameter controls the size/level of the heading, it must be a number between 1 and 6. The optional "class" parameter allows the use of a desired CSS class. The optional "anchor_name" parameter makes it possible to add an anchor to the heading. Usage:

```
<h[level=""][class=""] [anchor_name=""]>Example</h>
```

or

```
<header[level=""][class=""] [anchor_name=""]>Example</header>
```

By default, the specified levels are increased by one. In other words, a level 1 header in the XML block will become a level 2 header (H2) in the resulting HTML. The reason for this is because the H1 tag is reserved for the name / main title of the content object. The headings inside the XML block will thus become subheadings of the main title. This behavior can be changed by creating an override template for the "/content/datatype/view/ezxmltags/header.tpl" template (it can not be controlled from within an configuration file).

Bold text

Bold text can be achieved by using one of the following tags: "b", "bold" or "strong". The optional "class" parameter allows the use of a desired CSS class. Usage:

```
<b[class=""]>Boldtext.</b>
```

or

```
<bold[class=""]>Boldtext.</bold>
```

or

```
<strong[class=""]>Boldtext.</strong>
```

Italic text

Italic/emphasized text can be achieved by using one of the following tags: "i", "em" or "emphasize". The optional "class" parameter allows the use of a desired CSS class. Usage:

```
<i [class=""]>Emphasizedtext.</i>
```

or

```
<em [class=""]>Emphasizedtext.</em>
```

or

```
<emphasize [class=""]>Emphasizedtext.</emphasize>
```

Unformatted text

The "literal" tag can be used to output unformatted text, for example program source code, HTML code, XML content, etc. Everything that is inside a literal block will be rendered in the same way (character by character) as it is within the literal tags (the text will be output using the HTML PRE tags). The optional "class" parameter allows the use of a desired CSS class. Usage:

```
<literal[class=""]>Example<\literal>
```

Please note that in the example above, the slash is in the wrong way within the tag that terminates the literal block. This was done in order to make the tag appear on the documentation page (since we're using literal tags to make code blocks). In other words, it should be terminated with a frontslash instead of a backslash.

Lists

It is possible to create lists in the same way as in HTML by making use of the "ol", "ul" and "li" tags. The lists can be nested. The optional "class" parameter allows the use of a desired CSS class. The following examples demonstrate the usage of ordered and unordered lists.

Ordered lists

```
<ol>
<li>Element1</li>
<li>Element2</li>
<li>Element3</li>
</ol>
```

Unordered lists

```
<ul>
<li>Element1</li>
<li>Element2</li>
<li>Element3</li>
</ul>
```

Tables

Tables can be created in the same way as in HTML using "table", "tr", "th" and "td" tags. The tables can be nested. Usage:

```
<table[class=""] [border=""] [width=""]>  
...  
</table>
```

The "class", "border" and "width" parameters are optional. The "class" parameter can be used to assign a desired CSS class. The "border" parameter can be used to set a border (number of pixels). The "width" parameter can be used to control the table width (either 0-100% or number of pixels). Table content should be written according to normal HTML table syntax with "tr", "th" and "td" tags, see below.

Table rows

Table rows can be created in the same way as in HTML:

```
<tr>Tablerowcontentgoeshere.</tr>
```

Table headers

Table headers can be created in the same way as in HTML:

```
<th[class=""] [width=""] [rowspan=""] [colspan=""]>Example.</th>
```

All parameters are optional. The "class" parameter can be used to set the desired CSS class. The "width" parameter can be used to set the width (either as percentage or number of pixels). The "rowspan" and "colspan" parameters are the same as in HTML.

Table data/cell

Table data/cells can be created in the same way as in HTML:

```
<td[class=""] [width=""] [rowspan=""] [colspan=""]>Example.</td>
```

All parameters are optional. The "class" parameter can be used to set the desired CSS class. The "width" parameter can be used to set the width (either as percentage or number of pixels). The "rowspan" and "colspan" parameters are the same as in HTML.

Hyperlinks

Hyperlinks can be inserted by making use of the "a" or the "link" tags. Usage:

```
<a href="" [target=""] [class=""] [title=""] [id=""]>Example.</a>
```

or

```
<link href="" [target=""] [class=""] [title=""] [id=""]>Example.</link>
```

The "href" parameter is required and it must be set to a valid address (either external or internal). The "target" parameter can be used to determine how the target URL should be opened (inside the existing/active browser window/tab or within a new window/tab). The "class" parameter can be used to specify a CSS class that should be used when the link is rendered. The "title" parameter can be used to specify a short title text (will be shown when the pointer is hovering over the link). The "id" parameter is for assigning unique identifiers.

Internal links

It is possible to create internal links (to other nodes and objects) by making use of the "eznode://" and the "ezobject://" notation. The internal links will be created dynamically based on the node/object ID numbers. In other words, if a node is moved, the link(s) will point to the new location(s) and thus they will not be broken.

Link to a node

A link to a node can be created either by specifying the target node's ID number or the node path. The following examples demonstrate how an internal link to node number 128 can be created.

```
<a href="eznode://128">Example.</a>
```

or

```
<link href="eznode://128">Example.</link>
```

The following examples demonstrate how an internal link to a node located at "products/computers/example" can be created.

```
<a href="eznode://products/computers/example">Example.</a>
```

or

```
<link href="eznode://products/computers/example">Example.</link>
```

Link to an object

The following examples demonstrate how an internal link to object number 1024 can be created.

```
<a href="ezobject://1024">Example.</a>
```

or

```
<link href="ezobject://1024">Example.</link>
```

When object linking is used, the destination address will be generated using the main node assignment of the target object.

Anchors

The "anchor" tag makes it possible to insert HTML anchors inside the XML block. The inserted anchors will work like standard HTML anchors. Usage:

```
<anchorname="" />
```

The "name" parameter must be set to a unique identifier for the anchor. Anchors can be reached by appending the hash character (#) followed directly by the name of the anchor that the browser should jump to. Example: <http://www.example.com/hobbies#music>

Object embedding

The "embed" tag makes it possible to insert an arbitrary content object directly in the XML block. It can for example be used to embed images. Usage:

```
<embed href="" [class=""] [view=""] [align=""] [target=""] [size=""] [id=""] />
```

Parameter	Description	Required
href	The "href" parameter must be a valid link to either a node or an object using the same notation as for hyperlinks (for example "eznode://134", "eznode://path/to/some/node" and "ezobject://1024"). If the provided link is a link to a node, eZ Publish will use the object that is encapsulated by that node. In other words, in both cases it is the object that will be inserted (the node notation is just a wrapper).	Yes.
class	The "class" parameter makes it possible to specify a custom stylesheet that should be used. In the template, the specified stylesheet will be available in the \$classification variable.	No.
view	The "view" parameter makes it possible to specify the view mode that should be used when the object is rendered (for example "full", "line", etc.).	No.
align	The "align" parameter can be used to specify the positioning of the embedded object; possible values are "left", "center" and "right".	No.
target	The "target" parameter can be used to set the opening method (same browser tab/window or new browser tab/window) for the embedded	No.

	item (for example ”_self”, ”_blank”, etc.).	
size	The ”size” parameter can be used to set the image size that should be used when an image object is embedded (for example ”small”, ”medium”, ”large”, etc.). The available sizes are defined by image.ini.	No.
id	The ”id” parameter makes it possible to assign a unique ID which will be the ID attribute in the resulting HTML.	No.

Custom tags

In addition to the default tags described above, the "XML block" datatype makes it possible to use custom tags. A custom tag can be used both as a block or an inline element. Custom tags must be specified using the "AvailableCustomTags[]" array in the [CustomTagSettings] block within an override for the "content.ini" configuration file. When the XML is rendered, the contents of a custom tag will be replaced by a custom template. The name of the template must be specified using the "name" parameter. Example of usage:

```
<customname="template_name" [custom_parameter="value" [...] ]>  
The quick brown fox jumps over the lazy dog.  
</custom>
```

The custom tag in the example above will be replaced by a template called "template_name.tpl". This template must be located in the following directory within the current design: "/templates/content/datatype/view/ezxmltags/" (or one of the fallback designs). It is also possible to create an override template. The contents of the tag will be available in the "\$content" variable within the inserted template. The custom parameters are optional. When used, a custom parameter will be available as a template variable with the same name as it was specified in the tag itself.

Paragraphs

Paragraphs can be added by making use of either the 'p' or the 'paragraph' tag.

The optional 'class' parameter allows the use of a desired CSS class. If you do not specify the class parameter, the paragraph will be displayed in a natural way (without tags) in the administration interface. To create a non-classified paragraph, you can simply press 'Enter' key twice.

Usage:

```
<p [class=""]>Example</p>
```

or

```
<paragraph [class=""]>Example</paragraph>
```

By default, the system will use 'p' tag in the resulting XHTML code. This behavior can be changed by creating an override template for the '/content/datatype/view/ezxmltags/paragraph.tpl'.

5.2 Content classes

The classes are documented in the following sections:

- Content (page [345](#))
- Media (page [361](#))
- Users (page [368](#))

5.2.1 Content

Article (page 346)

Defines a structure for storing articles.

Comment (page 347)

Defines a structure for storing comments/feedback.

Company (page 348)

Defines a structure for storing information about companies.

Feedback form (page 349)

Defines a structure for feedback forms.

Folder (page 350)

Defines a structure for folders / information pages.

Forum (page 351)

Defines a structure for storing forums.

Forum reply (page 352)

Defines a structure for storing forum replies.

Forum topic (page 353)

Defines a structure for storing forum topics.

Gallery (page 354)

Defines a structure for storing image galleries.

Link (page 355)

Defines a structure for storing hyperlinks.

Person (page 356)

Defines a structure for storing information about people.

Poll (page 357)

Defines a structure for storing polls.

Product (page 358)

Defines a structure for storing information about products.

Review (page 359)

Defines a structure for storing product reviews.

Weblog (page 360)

Defines a structure for storing personal logs.

Article

Summary

Defines a structure for storing articles.

Properties

Name	Identifier	Container	Object name pattern
Article	article	Yes.	<short_title title>

Attributes

Name	Identifier	Datatype	R	S	C	T
Title	title	Text line	Yes.	Yes.	No.	Yes.
Short title	short_title	Text line	No.	Yes.	No.	Yes.
Author	author	Authors	No.	Yes.	No.	Yes.
Intro	intro	XML block	Yes.	Yes.	No.	Yes.
Body	body	XML block	Yes.	Yes.	No.	Yes.
Enable comments	enable_comments	Checkbox	No.	No.	No.	No.
Image	image	Object relation	No.	Yes.	No.	No.
Keywords	keywords	Keywords	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Comment

Summary

Defines a structure for storing comments/feedback.

Properties

Name	Identifier	Container	Object name pattern
Comment	comment	Yes.	<subject>

Attributes

Name	Identifier	Datatype	R	S	C	T
Subject	subject	Text line	Yes.	Yes.	No.	Yes.
Author	author	Text line	Yes.	Yes.	No.	Yes.
Message	message	Text block	Yes.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Company

Summary

Defines a structure for storing information about companies.

Properties

Name	Identifier	Container	Object name pattern
Company	company	Yes.	<company_name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Company name	company_name	Text line	Yes.	Yes.	No.	Yes.
Company number	company_number	Text line	No.	Yes.	No.	Yes.
Company address	company_address	Matrix	No.	Yes.	No.	Yes.
Logo	logo	Image	No.	Yes.	No.	Yes.
Additional information	additional_information	Text block	No.	Yes.	No.	Yes.
Contact information	contact_information	Matrix	No.	Yes.	No.	Yes.
Contacts	contacts	Object relations	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Feedback form

Summary

Defines a structure for feedback forms.

Properties

Name	Identifier	Container	Object name pattern
Feedback form	feedback_form	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Subject	subject	Text line	Yes.	Yes.	Yes.	Yes.
Message	message	Text block	Yes.	Yes.	Yes.	Yes.
Email	email	E-mail	Yes.	No.	Yes.	Yes.
Recipient	recipient	E-mail	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Folder

Summary

Defines a structure for folders / information pages.

Properties

Name	Identifier	Container	Object name pattern
Folder	folder	Yes.	<name short_name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Short name	short_name	Text line	Yes.	Yes.	No.	Yes.
Short description	short_description	XML block	No.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Show children	show_children	Checkbox	No.	No.	No.	No.

R Required

S Searchable

C Collector

T Translatable

Forum

Summary

Defines a structure for storing forums.

Properties

Name	Identifier	Container	Object name pattern
Forum	forum	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Forum reply

Summary

Defines a structure for storing forum replies.

Properties

Name	Identifier	Container	Object name pattern
Forum reply	forum_reply	Yes.	<subject>

Attributes

Name	Identifier	Datatype	R	S	C	T
Subject	subject	Text line	Yes.	Yes.	No.	Yes.
Message	message	Text block	Yes.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Forum topic

Summary

Defines a structure for storing forum topics.

Properties

Name	Identifier	Container	Object name pattern
Forum topic	forum_topic	Yes.	<subject>

Attributes

Name	Identifier	Datatype	R	S	C	T
Subject	subject	Text line	Yes.	Yes.	No.	Yes.
Message	message	Text block	Yes.	Yes.	No.	Yes.
Sticky	sticky	Checkbox	No.	No.	No.	Yes.
Notify me about updates	nofity_me	Subtree subscription	No.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Gallery

Summary

Defines a structure for storing image galleries.

Properties

Name	Identifier	Container	Object name pattern
Gallery	gallery	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Short description	short_description	XML block	No.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Image	image	Object relation	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Link

Summary

Defines a structure for storing hyperlinks.

Properties

Name	Identifier	Container	Object name pattern
Link	link	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Location	location	URL	No.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Person

Summary

Defines a structure for storing information about people.

Properties

Name	Identifier	Container	Object name pattern
Person	person	Yes.	<first_name> <last_name>

Attributes

Name	Identifier	Datatype	R	S	C	T
First name	first_name	Text line	Yes.	Yes.	No.	Yes.
Last name	last_name	Text line	Yes.	Yes.	No.	Yes.
Job title	job_title	Text line	No.	Yes.	No.	Yes.
Contact information	contact_information	Matrix	No.	Yes.	No.	Yes.
Picture	picture	Object relation	No.	Yes.	No.	Yes.
Comment	comment	XML block	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Poll

Summary

Defines a structure for storing polls.

Properties

Name	Identifier	Container	Object name pattern
Poll	poll	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Question	question	Option	Yes.	No.	Yes.	Yes.

R Required

S Searchable

C Collector

T Translatable

Product

Summary

Defines a structure for storing information about products.

Properties

Name	Identifier	Container	Object name pattern
Product	product	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Short description	short_description	XML block	No.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
Image	image	Object relation	No.	Yes.	No.	Yes.
Price	price	Price	No.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Review

Summary

Defines a structure for storing product reviews.

Properties

Name	Identifier	Container	Object name pattern
Review	review	Yes.	<summary>

Attributes

Name	Identifier	Datatype	R	S	C	T
Summary	summary	Text line	Yes.	Yes.	No.	Yes.
Author	author	Text line	No.	Yes.	No.	Yes.
Message	message	Text block	No.	Yes.	No.	Yes.
Rating	rating	Selection	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Weblog

Summary

Defines a structure for storing personal logs.

Properties

Name	Identifier	Container	Object name pattern
Weblog	weblog	Yes.	<title>

Attributes

Name	Identifier	Datatype	R	S	C	T
Title	title	Text line	Yes.	Yes.	No.	Yes.
Message	message	XML block	Yes.	Yes.	No.	Yes.
Enable comments	enable_comments	Checkbox	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

5.2.2 Media

File (page 362)

Defines a structure for storing binary files.

Flash (page 363)

Defines a structure for storing Macromedia Flash files.

Image (page 364)

Defines a structure for storing digital images.

QuickTime (page 365)

Defines a structure for storing Apple QuickTime files.

Real video (page 366)

Defines a structure for storing Real video files.

Windows media (page 367)

Defines a structure for storing ".avi" files.

File**Summary**

Defines a structure for storing binary files.

Properties

Name	Identifier	Container	Object name pattern
File	file	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
File	file	File	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Flash

Summary

Defines a structure for storing Macromedia Flash files.

Properties

Name	Identifier	Container	Object name pattern
Flash	flash	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
File	file	Media	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Image

Summary

Defines a structure for storing digital images.

Properties

Name	Identifier	Container	Object name pattern
Image	image	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Caption	caption	XML block	No.	Yes.	No.	Yes.
Image	image	Image	No.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

QuickTime

Summary

Defines a structure for storing Apple QuickTime files.

Properties

Name	Identifier	Container	Object name pattern
QuickTime	quicktime	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
File	file	Media	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Real video

Summary

Defines a structure for storing Real video files.

Properties

Name	Identifier	Container	Object name pattern
Real video	real_video	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
File	file	Media	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

Windows media

Summary

Defines a structure for storing ".avi" files.

Properties

Name	Identifier	Container	Object name pattern
Windows media	windows_media	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	XML block	No.	Yes.	No.	Yes.
File	file	Media	Yes.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

5.2.3 Users

User (page 369)

Defines a structure for storing user accounts.

User group (page 370)

Defines a structure for storing user groups.

User

Summary

Defines a structure for storing user accounts.

Properties

Name	Identifier	Container	Object name pattern
User	user	Yes.	<first_name> <last_name>

Attributes

Name	Identifier	Datatype	R	S	C	T
First name	first_name	Text line	Yes.	Yes.	No.	Yes.
Last name	last_name	Text line	Yes.	Yes.	No.	Yes.
User account	user_account	User account	Yes.	Yes.	No.	Yes.
Signature	signature	Text block	No.	Yes.	No.	Yes.
Image	image	Image	No.	No.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

User group

Summary

Defines a structure for storing user groups.

Properties

Name	Identifier	Container	Object name pattern
User group	user_group	Yes.	<name>

Attributes

Name	Identifier	Datatype	R	S	C	T
Name	name	Text line	Yes.	Yes.	No.	Yes.
Description	description	Text line	No.	Yes.	No.	Yes.

R Required

S Searchable

C Collector

T Translatable

5.3 Modules

class (page 373)

Provides views for managing classes, class groups, etc.

collaboration (page 392)

Provides an interface to the collaboration engine.

content (page 407)

Provides views for managing content (nodes, objects, searching, etc.)

error (page 527)

Provides an interface for error handling / reporting.

ezinfo (page 528)

Provides views for displaying information about eZ publish.

form (page 533)

Provides a view that generates an E-mail containing the data that was posted.

infocollector (page 536)

Provides views for managing collected information.

layout (page 541)

Provides a view that makes it possible to use alternative pagelayouts.

notification (page 546)

Provides an interface to the notification engine.

package (page 558)

Provides views for importing/exporting packages.

pdf (page 581)

Provides views for configuring PDF exports.

reference (page 585)

Provides a view for displaying documentation generated by Doxygen.

role (page 586)

Provides views for managing roles.

rss (page 594)

Provides views for managing RSS imports and exports.

search (page 600)

Provides a view that displays search statistics.

section (page 603)

Provides views for managing sections.

setup (page 617)

Provides the web based setup wizard.

shop (page 618)

Provides views for the webshop (basket, wish list, order list, etc.).

trigger (page 646)

Provides a view for managing workflow triggers.

url (page 649)

Provides views for managing the URLs stored in the database.

user (page 658)

Provides views for logging users in/out, password changing, etc.

workflow (page 683)

Provides views for managing workflows, workflow groups, workflow events, etc.

5.3.1 class

Summary

Provides views for managing classes, class groups, etc.

Description

This module provides several interfaces that can be used to view and manage the content classes and class groups that are present in the system. The views that the module provides are used by the "Class" section of the "Setup" part of the administration interface.

The module components are documented in the following sections:

- Fetch functions (page [374](#))
- Views (page [381](#))

Fetch functions

attribute_list (page 375)

Fetches the attributes of a class.

latest_list (page 376)

Fetches the most recently modified classes.

list (page 377)

Fetches a collection of classes.

override_template_list (page 379)

Fetches the override rules associated with a class.

attribute_list**Summary**

Fetches the attributes of a class.

Usage

```
fetch( 'class', 'attribute_list', hash( 'class_id', class_id ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The ID number of the target class.	Yes.

Returns

Array of ezcontentclassattribute (page 708) objects or FALSE.

Description

This function fetches the attributes of a class specified by the "class.id" parameter. The function returns an array of ezcontentclassattribute (page 708) objects or FALSE.

Examples**Example 1**

```
{def $attributes=fetch( 'class', 'attribute_list', hash( 'class_id', 13 ) )}

{foreach $attributes as $attribute}
    {$attribute.name|wash} <br />
{/foreach}
```

Outputs the names of the attributes that belong to class number 13.

latest_list

Summary

Fetches the most recently modified classes.

Usage

```
fetch( 'class', 'latest_list', hash( [ 'offset', offset ] ,
[ 'limit', limit ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	No.
limit	integer	The number of classes that should be fetched.	No.

Returns

Array of ezcontentclass (page 705) objects.

Description

This function fetches the most recently modified classes. The function returns an array of ezcontentclass (page 705) objects. The "offset" and "limit" parameters are optional and can be used to narrow down the result. If the "offset" and "limit" parameters are omitted, the function will simply return all the available classes.

Examples

Example 1

```
{def $classes=fetch( 'class', 'latest_list', hash( 'limit', 10 ) )}

{foreach $classes as $class}
    {$class.name|wash} <br />
{/foreach}
```

Outputs the names of the ten most recently modified classes.

list

Summary

Fetches a collection of classes.

Usage

```
fetch( 'class', 'list', hash( [ 'class_filter', array( class_id |  
class_identifier [, ...] ) ] ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The ID number of a desired class.	No.
class_identifier	string	The identifier of a desired class.	No.
...			

Returns

Array of ezcontentclass (page 705) objects or FALSE.

Description

This function fetches a collection of classes. The optional "class_filter" parameter can be used to fetch only a given set of classes. This parameter must be either an array of class ID numbers or an array of class identifier strings (in other words: mixing is not possible). If the "class_filter" array is omitted, all classes will be returned.

Examples

Example 1

```
{def $classes=fetch( 'class', 'list' )}  
  
{foreach $classes as $class}  
    {$class.name|wash} <br />  
{/foreach}
```

Outputs the names of all classes.

Example 2

```
{def $classes=fetch( 'class', 'list', hash( 'class_filter', array( 1, 2, 3 ) ) )}

{foreach $classes as $class}
    {$class.name|wash} <br />
{/foreach}
```

Outputs the names of class number 1, 2 and 3.

Example 3

```
{def $classes=fetch( 'class', 'list', hash( 'class_filter', array( 'folder',
'article' ) ) )}

{foreach $classes as $class}
    {$class.name|wash} <br />
{/foreach}
```

Outputs the names of the classes identified by the strings "folder" and "article".

override_template_list

Summary

Fetches the override rules associated with a class.

Usage

```
fetch( 'class', 'override_template_list', hash( 'class_id', class_id ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The ID number of the target class.	Yes.

Returns

An array of hashes containing information about the override rules.

Description

This function fetches the override rules that are associated with the class specified by the "class_id" parameter. The function returns an array of hashes. Each element of the returned array contains the following structure:

Attribute	Type	Description
siteaccess	string	The siteaccess that the override belongs to.
block	string	The name of the override block.
source	string	The path to the original template.
target	string	The path to the override template.

Examples

Example 1

```
{def $overrides=fetch( 'class', 'override_template_list', hash( 'class_id', 13 ) )}

{foreach $overrides as $override}
```

```
{$override.target} - {$override.source}) <br />  
{/foreach}
```

Outputs information about the overrides for class number 13.

Views

classlist (page 382)

Provides an interface for generating a class overview for a class group.

copy (page 383)

Provides an interface for copying a role.

down (page 384)

Provides an interface for moving an attribute to a lower position.

edit (page 385)

Provides an interface for editing a class.

groupedit (page 386)

Provides an interface for editing a class group.

grouplist (page 387)

Provides an interface for generating an overview of the class groups.

removeclass (page 388)

Provides an interface to the class removal mechanism.

removegroup (page 389)

Provides an interface to the class group removal mechanism.

up (page 390)

Provides an interface for moving an attribute to a higher position.

view (page 391)

Provides an interface for viewing a class.

classlist**Summary**

Provides an interface for generating a class overview for a class group.

copy

Summary

Provides an interface for copying a role.

down

Summary

Provides an interface for moving an attribute to a lower position.

edit

Summary

Provides an interface for editing a class.

groupedit**Summary**

Provides an interface for editing a class group.

grouplist**Summary**

Provides an interface for generating an overview of the class groups.

removeclass**Summary**

Provides an interface to the class removal mechanism.

removegroup**Summary**

Provides an interface to the class group removal mechanism.

up

Summary

Provides an interface for moving an attribute to a higher position.

view

Summary

Provides an interface for viewing a class.

5.3.2 collaboration

Summary

Provides an interface to the collaboration engine.

Description

This module provides an interface to the collaboration engine inside the eZ publish kernel. The administration interface makes use of the views that the module provides in order to allow the management of collaboration items. Although possible, it isn't common to use these views when building a website (unless there is a need to replicate the collaboration management functionality of the administration interface).

The module components are documented in the following sections:

- Fetch functions (page [393](#))
- Views (page [402](#))

Fetch functions

group_tree (page 394)

Not documented yet.

item_count (page 395)

Not documented yet.

item_list (page 396)

Not documented yet.

message_list (page 397)

Not documented yet.

participant (page 398)

Not documented yet.

participant_list (page 399)

Not documented yet.

participant_map (page 400)

Not documented yet.

tree_count (page 401)

Not documented yet.

group_tree**Summary**

Not documented yet.

item_count**Summary**

Not documented yet.

item_list**Summary**

Not documented yet.

message_list**Summary**

Not documented yet.

participant

Summary

Not documented yet.

participant_list**Summary**

Not documented yet.

participant_map**Summary**

Not documented yet.

tree_count

Summary

Not documented yet.

Views

action (page 403)

Not documented yet.

group (page 404)

Not documented yet.

item (page 405)

Not documented yet.

view (page 406)

Not documented yet.

action

Summary

Not documented yet.

group

Summary

Not documented yet.

item

Summary

Not documented yet.

view

Summary

Not documented yet.

5.3.3 content

Summary

Provides views for managing content (nodes, objects, searching, etc.)

Description

This module provides an interface to the content engine that is built into eZ Publish. It is the most important and most commonly utilized module. It provides views that make it possible to display and edit the contents of objects, manage nodes in the content tree, searching, translation, etc. A typical eZ Publish site uses many of the views and the fetch functions that this module provides.

The module components are documented in the following sections:

- Fetch functions (page [408](#))
- Views (page [494](#))

Fetch functions

access (page 411)

Checks if the current user has access to a given function.

bookmarks (page 414)

Fetches the bookmarks of the current user.

calendar (page 416)

Not documented yet.

can_instantiate_classes (page 417)

Checks if the current user is allowed to create nodes.

can_instantiate_class_list (page 419)

Fetches the classes that the current user can create objects of.

class (page 421)

Fetches a content class.

class_attribute (page 422)

Fetches an attribute of a content class.

class_attribute_list (page 423)

Fetches the attributes of a class.

collected_info_collection (page 425)

Fetches an information collection.

collected_info_count (page 426)

Fetches the number of collections that match a certain criteria.

collected_info_count_list (page 428)

Fetches the number of times different values were collected.

contentobject_attributes (page 429)

Fetches the attributes of an object's version (and translation).

draft_count (page 431)

Fetches the number of drafts that belong to the current user.

draft_version_list (page 432)

Fetches the drafts that belong to the current user.

keyword (page 433)

Fetches nodes that use keywords starting with a given sequence.

keyword_count (page 435)

Fetches the number of nodes that use certain keywords.

list (page 436)

Fetches the children of a node or a collection of nodes.

list_count (page 451)

Fetches the number of children of a node.

locale_list (page 453)

Fetches the available locales.

navigation_part (page 454)

Fetches information about a navigation part.

navigation_parts (page 455)

Fetches all available navigation parts.

node (page 456)

Fetches a node (identified by either an ID number or a path).

non_translation_list (page 458)

Fetches locales that a version of an object may be translated into.

object (page 459)

Fetches a content object (specified by an ID number).

object_by_attribute (page 460)

DEPRECATED

object_count_by_user_id (page 461)

Fetches the number of objects (of a class) created by a user.

pending_count (page 462)

Fetches the number of pending objects for the current user.

pending_list (page 463)

Fetches the pending objects for the current user.

recent (page 464)

Fetches nodes where the current user recently published something.

related_objects (page 465)

Fetches related objects.

related_objects_count (page 468)

Fetches the number of related objects.

reverse_related_objects (page 469)

Fetches reverse relates objects.

reverse_related_objects_count (page 472)

Fetches the number of reverse related objects.

same_classattribute_node (page 473)

Fetches nodes containing attributes that match a certain value.

search (page 475)

Fetches nodes containing data that match a certain criteria.

section_list (page 478)

Fetches the available sections.

tipafriend_top_list (page 479)

Fetches the most popular (most tipped) nodes.

translation_list (page 480)

Fetches the locales that can be used to translate objects.

trash_count (page 481)

Fetches the number of objects that are in the trash.

trash_object_list (page 482)

Fetches the objects that are in the trash.

tree (page 483)

Fetches the children of a node recursively.

tree_count (page 486)

Fetches the number of children of a node recursively.

version (page 488)

Fetches a specific version of an object.

version_count (page 489)

Fetches the number of versions of a content object.

version_list (page 490)

Fetches all the versions of a content object.

view_top_list (page 492)

Fetches the most popular (most viewed) nodes.

access**Summary**

Checks if the current user has access to a given function.

Usage

```
fetch( 'content', 'access',
      hash( 'access',                               access,
            'contentobject',                         contentobject,
            [ 'contentclass_id',                     contentclass_id,       ]
            [ 'parent_contentclass_id', parent_contentclass_id ] ) )
```

Parameters

Name	Type	Description	Required
access	string	The desired access method (see below).	Yes.
contentobject	object	The target/location (either an object or a node).	Yes.
contentclass_id	integer, string	The ID number or identifier of the class that should be included in the check.	No.
parent_contentclass_id	integer, string	The parent node's class ID number or identifier that should be included in the check.	No.

Returns

TRUE if access is allowed, FALSE otherwise.

Description

This function makes it possible to find out if the current user has access (read, edit, create, remove, etc.) to a given content object or a content node. The optional parameters "contentclass_id" and "parent_content_class_id" can be used to finetune the checking. These parameters are compatible with both class ID numbers and class identifier strings. The function supports checking for the following access methods:

- bookmark
- create
- edit

- move
- read
- remove
- pdf
- restore
- translate
- versionread

When checking "create" access and the "contentclass_id" is not specified, the function will return TRUE as long as there is a create access for the given object. However, the user could still not be allowed to create a specific class.

Examples

Example 1

```
{def $test=fetch( 'content', 'access',
                  hash( 'access', 'read',
                        'contentobject', $node ) )}

{if $test}
    The current user has read access to the given node.
{else}
    The current user does not have read access to the given node.
{/if}
```

Checks if the content node represented by \$node is can be read by the current user.

Example 2

```
{def $test=fetch( 'content', 'access',
                  hash( 'access', 'create',
                        'contentobject', $node,
                        'contentclass_id', 'folder' ) )}

{if $test}
    The current user can create a folder below the given node.
{else}
    The current user can not create a folder below the given node.
{/if}
```

Checks if the current user can create an instance of a given class below the specified node.

bookmarks

Summary

Fetches the bookmarks of the current user.

Usage

```
fetch( 'content', 'bookmarks', hash( [ 'offset', offset, ]
[ 'limit', limit ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	Number of bookmarks to skip.	No.
limit	integer	Maximum number of bookmarks to fetch.	No.

Returns

An array of ezcontentbrowsebookmark (page 703) objects.

Description

Fetches the bookmarks of the current user and returns an array of ezcontentbrowsebookmark (page 703) objects. The resulting array starts with the most recently added bookmark.

Examples

Example 1

```
{def $bookmarks=fetch( 'content', 'bookmarks' )}

{foreach $bookmarks as $bookmark}

    <a href={$bookmark.node.url_alias|ezurl}>{$bookmark.name|wash}</a> <br />

{/foreach}
```

Outputs all the bookmarks (as links) for the current user.

Example 2

```
{def $bookmarks=fetch( 'content', 'bookmarks' )}

{foreach $bookmarks as $bookmark}

    <a href={$bookmark.node.url_alias|ezurl}>{$bookmark.name|wash}</a> <br />

{/foreach}
```

Outputs the five most recently added bookmarks (as links) for the current user.

calendar**Summary**

Not documented yet.

can_instantiate_classes

Summary

Checks if the current user is allowed to create nodes.

Usage

```
fetch( 'content', 'can_instantiate_classes' [, hash( 'parent_node',
parent_node ) ] )
```

Parameters

Name	Type	Description	Required
parent_node	object	Node to check if user can create new object in	No.

Returns

TRUE if the current user is allowed to create nodes, FALSE otherwise.

Description

This fetch function checks if the current user is allowed to create nodes either below the current position within the tree or below a given node (specified by the "parent_node" parameter). The function will return TRUE if the current user is allowed to create nodes, otherwise FALSE will be returned.

Examples

Example 1

```
{if fetch( 'content', 'can_instantiate_classes' )}
    The current user can create nodes below the current node.
{else}
    The current user can not create nodes below the current node.
{/if}
```

Example 2

```
{def $target=fetch( 'content', 'node', hash( 'node_id', 64 )}

{if fetch( 'content', 'can_instantiate_classes', hash( 'parent_node', $target
)}
    The current user can create nodes below node number 64.
{else}
    The current user can not create nodes below node number 64.
{/if}
```

can_instantiate_class_list

Summary

Fetches the classes that the current user can create objects of.

Usage

```
fetch( 'content', 'can_instantiate_class_list', hash( [ 'group_id',
group_id      ],
parent_node ] )
```

Parameters

Name	Type	Description	Required
group_id	integer	The ID number of a class group to fetch classes from.	No.
parent_node	object	Alternative parent node.	No.

Returns

Array of ezcontentclass (page 705) objects or FALSE.

Description

This function fetches a list of classes that the current user is allowed to create objects from. If no parameters are given, the class list will be generated based on the current node and classes from all class groups. The "group_id" parameter can be used to instruct the system to only fetch classes from a certain class group. The "parent_node" parameter can be used to instruct the system to check which classes the current user is allowed to instantiate below a certain node instead of the current node. The function returns an array of ezcontentclass (page 705) objects or FALSE / empty array if the current user can not create instances of any class.

Examples

Example 1

```
{def $classes=fetch( 'content', 'can_instantiate_class_list' )}

{foreach $classes as $class}
    {$class.name} <br />
{/foreach}
```

Outputs the name of all classes that the current user is allowed to create below the current node.

Example 2

```
{def $classes=fetch( 'content', 'can_instantiate_class_list', hash(  
'group_id', 3 )}  
  
{foreach $classes as $class}  
    {$class.name} <br />  
{/foreach}
```

Outputs the name of classes belonging to class group number 3 that the current user is allowed to create below the current node.

class

Summary

Fetches a content class.

Usage

```
fetch( 'content', 'class', hash( 'class_id', class_id ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The identifier or ID number of the of class that should be fetched.	Yes.

Returns

The specified class as an ezcontentclass (page 705) object or FALSE.

Description

This function fetches the class specified by the "class_id" parameter and returns it as an ezcontentclass (page 705) object. The function will return FALSE if the specified class is unavailable.

Examples

Example 1

```
{def $class=fetch( 'content', 'class', hash( 'class_id', 13 ) )}  
{$class.name|wash}
```

Outputs name of class number 13.

class_attribute

Summary

Fetches an attribute of a content class.

Usage

```
fetch( 'content', 'class_attribute', hash( 'attribute_id', attribute_id,
                                         [ 'version_id', version_id ] ) )
```

Parameters

Name	Type	Description	Required
attribute_id	integer	The ID number of the class attribute that should be fetched.	Yes.
version_id	integer	The status/version of the class.	No.

Returns

The specified class attribute as an ezcontentclassattribute object or FALSE.

Description

This function fetches the class attribute specified by the "attribute_id" parameter and returns it as an ezcontentclassattribute (page 708) object. The function will return FALSE if the specified class attribute is unavailable. The optional "version_id" parameter can be used to fetch the specified class attribute from a certain version of a class. The following versions / status codes can be used:

- 0: defined
- 1: temporary
- 2: modified

Examples

Example 1

```
{def$class_attribute=fetch('content','class_attribute',hash('attribute_id',231))} {$class_attribute.name|wash}
```

Outputs the name of class attribute number 231.

class_attribute_list

Summary

Fetches the attributes of a class.

Usage

```
fetch( 'content', 'class_attribute_list',
       hash( 'class_id', class_id,
             [ 'version_id', version_id ] ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The ID number of the target class.	Yes.
version_id	integer	The version/status of the class.	No.

Returns

Array of ezcontentclassattribute (page 708) objects or FALSE.

Description

This function fetches the attributes of a class. The ID number of the target class must be specified using the "class_id" parameter. The function returns an array of ezcontentclassattribute (page 708) objects or FALSE if the specified class attribute is unavailable. The optional "version_id" parameter can be used to fetch class attributes from a certain version of a class. The following versions / status codes can be used:

- 0: defined
- 1: temporary
- 2: modified

Examples

Example 1

```
{def $attributes=fetch( 'content', 'class_attribute_list', hash( 'class_id',
13 ) )}

{foreach $attributes as $attribute}
```

```
{$attribute.name|wash} <br />
{/foreach}
```

Outputs the names of the attributes that belong to class number 13.

collected_info_collection

Summary

Fetches an information collection.

Usage

```
fetch( 'content', 'collected_info_collection',
       hash( 'collection_id',    collection_id,
             'contentobject_id', contentobject_id ) )
```

Parameters

Name	Type	Description	Required
collection_id	integer	The ID number of the collection that should be fetched.	Yes.
contentobject_id	integer	The ID number of the object that should be fetched.	Yes.

Returns

An ezinformationcollection (page 736) object or FALSE.

Description

This function fetches an information collection. Both the ID number of the collection and the contentobject must be provided. The function returns an ezinformationcollection (page 736) object or FALSE.

Examples

Example 1

```
{def $collection=fetch( 'content', 'collected_info_collection',
                      hash( 'collection_id',    123,
                            'contentobject_id', 456 ) )}

{foreach $collection.attributes as $attribute}
  {$attribute.contentclass_attribute_name} <br />
{/foreach}
```

Outputs the attributes for the 123rd information collection for object number 456.

collected_info_count

Summary

Fetches the number of collections that match a certain criteria.

Usage

```
fetch( 'content', 'collected_info_count',
       hash( [ 'object_attribute_id', object_attribute_id,
              [ 'object_id',          object_id,
                [ 'value',            value
                  ] ] ) )
```

Parameters

Name	Type	Description	Required
object_attribute_id	integer	The ID number of the target object attribute.	No.
object_id	integer	The ID number of the target content object.	No.
value	integer	Value filtering on the attribute level.	No.

Returns

The number of collections (as an integer).

Description

This function counts the number of collections based on the provided parameters. An object's ID number (using the "object_id" parameter) or an object attribute's ID number (using the "object_attribute_id" parameter) must be specified. In addition, it is possible to filter out collections that match a certain value. This is typically useful when it comes to counting the number of times a specific value was submitted to a poll. If the "value" parameter is used then the "object_attribute_id" parameter must also be provided. The function returns a positive integer if the system is able to find collections matching the given parameters; if not, zero will be returned.

Examples

Example 1

```
{def $collections=fetch( 'content', 'collected_info_count',
                        hash( 'object_attribute_id', 42,
                              'object_id',           20,
```

```
        'value' ,  
{$collections}  
    1 ) )}
```

Outputs the number of times attribute #42 for object number 20 has collected "1".

collected_info_count_list

Summary

Fetches the number of times different values were collected.

Usage

```
fetch( 'content', 'collected_info_count_list', hash( 'object_attribute_id',
object_attribute_id ) )
```

Parameters

Name	Type	Description	Required
object_attribute_id	integer	The ID number of the target object attribute.	Yes.

Returns

An array of integers representing a count for every value.

Description

This function calculates and returns the sum of collected values (integers). It was developed for the poll feature.

Examples

Example 1

```
{def $counts=fetch( 'content', 'collected_info_count_list',
                    hash( 'object_attribute_id', 1024 ) )}

{foreach $counts as $count}
    {$count} <br />
{/foreach}
```

Outputs the number of times different values were collected.

contentobject_attributes

Summary

Fetches the attributes of an object's version (and translation).

Usage

```
fetch( 'content', 'contentobject_attributes',
       hash( 'version',      version,
             [ 'language_code', language_code ] ) )
```

Parameters

Name	Type	Description	Required
version	object	The target version (must be an ezcontentobjectversion object).	Yes.
language_code	string	The language code.	No.

Returns

An array of ezcontentobjectattribute (page 717) objects or FALSE.

Description

This function fetches the attributes that belong to a certain version. The version must be provided (as an ezcontentobjectversion (page 725)) using the "version" parameter. The "language_code" parameter is optional and can be used to get the attributes that belong to a specific translation. The function returns an array of ezcontentobjectattribute (page 717) objects or FALSE if something went wrong.

Examples

Example 1

```
{def $object=fetch( 'content', 'object', hash( 'object_id', 14 ) )
    $attributes=fetch( 'content', 'contentobject_attributes',
                      hash( 'version', $object.current ) )}

{foreach $attributes as $attribute}
    {$attribute.data_type_string} <br />
{/foreach}
```

Outputs the names of the datatypes that are used by the different attributes within the current version of object number 14.

draft_count**Summary**

Fetches the number of drafts that belong to the current user.

Usage

```
fetch( 'content', 'draft_count' )
```

Returns

The number of drafts (as an integer) that belong to the current user.

Description

This function fetches the number of drafts that belong to the current user and returns it as an integer.

Examples**Example 1**

```
{def $drafts=fetch( 'content', 'draft_count' )}  
The current user has {$drafts} drafts.
```

Outputs the number of drafts that belong to the current user.

draft_version_list

Summary

Fetches the drafts that belong to the current user.

Usage

```
fetch( 'content', 'draft_version_list',
       hash( [ 'offset', offset,
               [ 'limit', limit ] ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	No.
limit	integer	The number of drafts/versions that should be fetched.	No.

Returns

An array of ezcontentobjectversion (page 725) objects or FALSE.

Description

This function fetches the drafts that belong to the current user. The optional parameters "offset" and "limit" can be used to limit the result. The function returns an array of ezcontentobjectversion (page 725) objects. If no drafts can be found or if something goes wrong, the function returns FALSE.

Examples

Example 1

```
{def $drafts=fetch( 'content', 'draft_version_list' )}

{foreach $drafts as $draft}
    {$draft.id}: {$draft.name} <br />
{/foreach}
```

Outputs the ID numbers and the names of all drafts that belong to the current user.

keyword**Summary**

Fetches nodes that use keywords starting with a given sequence.

Usage

```
fetch( 'content', 'keyword', hash( 'alphabet', alphabet,
                                [ 'classid', classid, ]
                                [ 'limit', limit, ]
                                [ 'offset', offset, ] ) )
```

Parameters

Name	Type	Description	Required
alphabet	string	The sequence that should be matched.	Yes.
classid	array	Filtering: the ID number of the class or an array of the ID numbers.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of elements that should be returned.	No.

Returns

An array of hashes (see below) or FALSE.

Description

This function fetches nodes that encapsulate objects which make use of certain keywords. The keyword must be provided using the "alphabet" parameter. This parameter can be a letter, a part of a word or an entire word - the function will look for keywords that start with the specified sequence. By default, the function will fetch nodes that encapsulate objects of all types. However, it is possible to only fetch objects of a certain type, this can be achieved by using the optional "classid" parameter. The "offset" and "limit" parameters can be used to limit the result. The function returns an array of hashes. Each hash consists of the following elements:

Key	Type	Description
keyword	string	The keyword that was matched.
link_object	object	The node (as a ezcontentobjecttreenode (page 721) object) that encapsulates an object which uses the matched

keyword.

If no match is found, the function will return FALSE.

Examples

Example 1

```
{def $list=fetch( 'content', 'keyword',
                  hash( alphabet, 'computer',
                        classid, 3 ) )}

{foreach $list as $element}
  {$element.link_object.name|wash} ({$element.keyword|wash}) <br />
{/foreach}
```

Outputs the names of nodes that encapsulate objects which make use of keywords starting with the string "computer". In addition, the matched keywords are also printed.

Example 2

```
{def $list=fetch( 'content', 'keyword',
                  hash( alphabet, 'computer',
                        classid, array( 1, 3 ) ) )}
```

Only nodes that encapsulate objects of the specified two classes will be fetched.

keyword_count

Summary

Fetches the number of nodes that use certain keywords.

Usage

```
fetch( 'content', 'keyword_count', hash( 'alphabet', alphabet,
                                         [ 'classid', classid ] ) )
```

Parameters

Name	Type	Description	Required
alphabet	string	The sequence that should be matched.	Yes.
classid	integer	Filtering: the ID number of the class.	No.

Returns

The number of matching nodes (as an integer).

Description

This function returns the number of nodes that encapsulate objects which make use of certain keywords. The keyword must be provided using the "alphabet" parameter. This parameter can be a letter, a part of a word or an entire word - the function will look for keywords that start with the specified sequence. By default, the function will count nodes that encapsulate objects of all types. However, it is possible to only count objects of a certain type, this can be achieved by using the optional "classid" parameter. The function returns an integer.

Examples

Example 1

```
{def $count=fetch( 'content', 'keyword_count',
                   hash( 'alphabet', 'computer' ) )}
```

There are {\$count} number of nodes using keywords starting with "computer".

Outputs the number of nodes that encapsulate objects which use keywords starting with "computer".

list

Summary

Fetches the children of a node or a collection of nodes.

Usage

```
fetch( 'content', 'list',
hash( 'parent_node_id', parent_node_id,
      [ 'sort_by', sort_by, ],
      [ 'offset', offset, ],
      [ 'limit', limit, ],
      [ 'attribute_filter', attribute_filter, ],
      [ 'extended_attribute_filter', extended_attribute_filter, ],
      [ 'class_filter_type', class_filter_type, ],
      [ 'class_filter_array', class_filter_array, ],
      [ 'only_translated', only_translated, ],
      [ 'language', language, ],
      [ 'main_node_only', main_node_only, ],
      [ 'as_object', as_object, ],
      [ 'depth', depth, ],
      [ 'limitation', limitation ],
      [ 'ignore_visibility', ignore_visibility ] ) )
```

Parameters

Name	Type	Description	Required
parent_node_id	mixed	The ID number(s) of the parent node(s).	Yes.
sort_by	array	The sorting mechanism that should be used.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of nodes that should be fetched.	No.
attribute_filter	mixed	The attribute level filter logic.	No.
extended_attribute_filter	mixed	The extended attribute level filter logic.	No.
class_filter_type	string	The type of class filtering (include/exclude).	No.
class_filter_array	array	The classes that should be filtered.	No.
group_by	array	DEPRECATED	No.
only_translated	boolean	Translation filtering (on/off).	No.
language	string	The translations that should be fetched.	No.
main_node_only	boolean	Type of nodes that should be fetched (all or main nodes only).	No.
depth	integer	The maximum level of depth that should be explored (1 by default).	No.
limitation	array	Limitation array (empty array = access override).	No.
ignore_visibility	boolean	Makes it possible to fetch hidden nodes.	No.
as_object	boolean	If TRUE (or omitted), an array of "ezcontentobjecttreenode" objects will be fetched. Otherwise, an array of arrays will be returned.	No.

Returns

An array of ezcontentobjecttreenode (page 721) objects or FALSE.

Description

This function fetches the children of a single node or a collection of nodes. The parent node(s) must be specified using the "parent_node_id" parameter. This parameter can either be the ID number of a single node or an array containing several node ID numbers. The function will fetch the nodes that are directly under the specified parent node(s). The collection is returned as an array of ezcontentobjecttreenode (page 721) objects. If no nodes are found or if an error occurs,

the function will return FALSE. The optional "group_by" parameter (grouping on a date/time basis) is deprecated since this functionality is not supported by Oracle and PostgreSQL.

Sorting

The "sort_by" parameter makes it possible to sort the result in different ways. This parameter must be provided as an array. The first element of the array must be the desired sorting method. The second element of the array must be the sorting direction, it can be either true() or false() - ascending or descending. The following table shows the sorting methods that can be used.

Sorting method	Description
attribute	The nodes are sorted according to the value of a specific attribute.
class_identifier	The nodes are sorted by the class identifiers of the objects.
class_name	The nodes are sorted by the class names of the objects.
depth	The nodes are sorted by their depth in the content tree.
modified	The nodes are sorted by the modification time of the objects.
name	The nodes are sorted by the names of the objects.
path	The nodes are sorted by their node ID path strings (/1/2/43/56).
path_string	The nodes are sorted by their virtual path strings (/company/about).
priority	The nodes are sorted by their priority.
published	The nodes are sorted by the creation time of the objects.
section	The nodes are sorted by the section IDs of the objects.

It is possible to combine different sorting methods. For example, it is possible to sort the nodes alphabetically and by their publish date/time at the same time.

Sorting on the attribute level is supported for the following datatypes:

- Checkbox
- Date
- Date and time
- E-mail
- Integer
- Selection (will not work when used as multiple selector)

- Text line
- Time

The syntax for doing attribute level sorting is almost the same as for normal sorting. The only difference is that the ID number or the identifier of the target attribute must be specified. If the identifier is used then both the identifier of the class and the attribute must be specified (separated by a slash, like this: "my_class/my_attribute"). Attribute sorting can only be used if the returned collection contains the same type of nodes.

Fetching subsets

By making use of the "offset" and "limit" parameters, it is possible to fetch only a subset of the collection that would have been returned if these parameters were omitted. While the "offset" parameter defines the start of the subset, the "limit" parameter defines the length (number of elements/nodes) of the subset. These parameters are processed at the final stage. In other words, it is possible to do advanced sorting/filtering and grab only a specific chunk of from the sorted/filtered/etc. collection.

Class filtering

The class filter mechanism is controlled by the "class_filter_type" and "class_filter_array" parameters. The "class_filter_type" parameter tells eZ Publish to include or exclude specific node types. This parameter must be either "include" or "exclude". It is not possible to include one set and to exclude another at the same time. The value of the "class_filter_array" parameter specifies the type of nodes that should be included or excluded. This parameter must be an array of class ID numbers or class identifier strings.

Attribute filtering

The attribute filter mechanism is controlled by the "attribute_filter" parameter. Attribute filtering makes it possible to fetch a set of nodes where an attribute (or several attributes) contains some specific data. Filtering on the attribute level is supported for the following datatypes:

- Checkbox
- Date
- Date and time
- E-mail
- Integer
- Object relation

- Selection (will not work when used as multiple selector)
- Text line
- Time

The "attribute_filter" parameter must be an array. The first element may be set to either "and" or "or" - this controls how the matching specified in the upcoming elements should be carried out. If this parameter is omitted, the system will default to "and". The rest of the elements are arrays, each array specifies a match. The elements in this array are:

1. Attribute ID
2. Match type
3. Match value

The attribute ID can be specified either as the ID number of the attribute or as a string that contains the identifier of the class and the identifier of the attribute separated by a slash. The ID number of the attribute or the identifier of the class and the attribute can be found when viewing/editing content classes. The match type tells how eZ Publish should try to match the values. The match value is the data that should be matched. The following match types can be used:

Type	Description
=	Equal
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
in	Matches one of the array values
not_in	Does not match any of the array values
between	Is a part of the range determined by the start and end values
not_between	Is outside the range determined by the start and end values
like	Matches the wildcard string (the wildcard character is '?')

It is not possible to filter on attributes of different classes, for example it is not possible to filter on both "article/show_comments" and "folder/show_comments" in the same filter. But instead of the class/attribute syntax, you can also use "published" and "priority" as "AttributeID". It is also possible to use "owner" as "AttributeID" which refers to the owner's identifier of the content object. This can for instance be used to find all objects of a given user.

Extended attribute filtering

The "extended_attribute_filter" parameter can be used to do advanced attribute filtering. It is for expert/experienced users. This mechanism makes it possible to introduce additional SQL conditions within the fetch. It allows filtering based on values in custom tables and/or special filtering within the "ezcontentobject_attribute" table.

The mechanism must be configured using a configuration override for "extendedattribute-filter.ini". This file allows the site administrator to set up different custom filters. Each filter must be a named collection of configuration settings. The name of a filter is the name of the configuration block under which the filter's settings are defined. The following text shows an example that demonstrates the setup of a filter called "MyFilter".

```
#The name of the filter.  
[MyFilter]  
  
#The name of the extension where the filtering code is defined.  
ExtensionName=myextension  
  
#The name of the filter class.  
ClassName=eZMyExtendedFilter  
  
#The name of the method which is called to generate the SQL parts.  
MethodName=createSqlParts  
  
#The file which should be included (extension/  
myextension will automatically be prepended).  
FileName=classes/ezmyextendedfilter.php
```

The function defined by the "MethodName" setting must return an associative array containing two strings:

```
array( 'tables' => '...', 'joins' => '...' );
```

The contents of "tables" must start with a comma. The rest of the string should contain a comma separated list of other tables that should be included in the query. The contents of "joins" will be added inside the "WHERE" section of the query. This string must start with a space and contain an "AND" and an additional space at the end.

The example below shows a solution that filters the content based on values within an additional/custom table. This table is joined with the node table (by object ID and object version). The additional/custom table would most likely be maintained by a special/custom datatype.

```
class eZMyExtendedFilter  
{  
    /*!  
     Constructor
```

```
/*
function eZMyExtendedFilter()
{
    // Empty...
}

function createSqlParts( $params )
{
    $sqlTables= ', ezmyfiltertable ';

    $sqlJoins = ' ezcontentobject_tree.contentobject_id =
ezmyfiltertable.contentobject_id AND
ezcontentobject_tree.contentobject_version = ezmyfiltertable.version AND ';

    if ( isset( $params['value1'] ) )
    {
        $value1 = $params['value1'];
    }
    else
    {
        $value1 = "10";
    }
    if ( isset( $params['value2'] ) )
    {
        $value2 = $params['value2'];
    }
    else
    {
        $value2 = "10";
    }

    $sqlCondArray = array();

    $sqlCondArray[] = 'ezmyfiltertable.my_cond1 = ' . $value1;
    $sqlCondArray[] = 'ezmyfiltertable.my_cond1 = ' . $value2;

    $sqlCond = implode( ' or ', $sqlCondArray );

    $sqlCond = ' ( ' . $sqlCond . ' ) AND ' . $sqlJoins . ' ';

    return array( 'tables' => $sqlTables, 'joins' => $sqlCond );
}
```

The following template code shows how the extended attribute filter (see the PHP code above) can be used within a fetch.

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 2,
        'sort_by', array( 'priority', false() ),
        'limit', 15,
        'extended_attribute_filter',
            hash( 'id', 'MyFilter',
                'params', hash( 'value1', 15,
                    'value2', 30 ) ),
        'depth', 10,
    'main_node_only', true() )}
```

There is an additional example at the bottom of this page: Example 19.

Fetching main nodes only

The "main_node_only" parameter can be used to tell eZ Publish that it should only fetch main nodes (nodes that are main locations for content objects). This parameter is optional and can be set to either true() or false(). When set to true(), the fetch function will only return main nodes. This functionality can be used to avoid duplicates (different nodes but same objects) in the result.

Fetching translated objects only

In some multi-language scenarios, not all objects are translated into other/all languages. When the "only_translated" parameter is set to "true()" the system will only fetch objects that have been translated into the language used by the siteaccess (the current language). The "language" parameter can be used to control which translations that should be fetched. The value of this parameter must be a valid locale string, for example "ger-DE".

Access override

The "limitation" parameter makes it possible to instruct the system to use an alternate set of access limitations instead of the ones that belong to the current user. This is typically useful when there is a need to fetch nodes that the current user does not have access to. All permission checking can be skipped by providing an empty array. It is also possible to provide arrays that dictate the access limitations of other users and/or virtual sets of limitations. The array follows an internal structure that will be documented in the future.

Visibility override

The "ignore_visibility" parameter makes it possible to fetch hidden nodes. It can be set to either "true()" or "false()". If set to "true()", the fetch will ignore the visibility flags of nodes and thus it will return all nodes regardless of their visibility status. In other words, this parameter overrides the "ShowHiddenNodes" (page 1342) configuration directive for a specific fetch operation in a template.

Examples

Example 1

```
{def $nodes=fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
        'depth', 3 ) )}

{foreach $nodes as $node}
    {$node.name|wash} <br />
{/foreach}
```

This example demonstrates how to fetch nodes which are until the third level under the parent node number 42. Outputs the names of all these nodes.

Example 2

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
        'sort_by', array( 'published', false() ) ) }
```

This example demonstrates how to fetch all nodes that are the children of node number 42. The nodes are sorted by the time they were created; the most recently created node will be the first element in the collection.

Example 3

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
        'sort_by', array( array( 'name', false() ),
            array( 'published', false() ) ) ) }
```

This example demonstrates how to combine different sorting methods. The fetch function will fetch the child nodes of node number 42. The fetched nodes will be sorted by their name and the time they were created.

Example 4

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
          'sort_by',           array( 'attribute',
                                         false(),
                                         107 ) ) )}
```

This example demonstrates attribute sorting by using the ID number of the attribute.

Example 5

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
          'sort_by',           array( 'attribute',
                                         false(),
                                         'person/last_name' ) ) )}
```

This example demonstrates attribute sorting by using the string notation to pinpoint the identifier of the class and the attribute.

Example 6

```
{fetch( 'content', 'list',
    hash( 'parent_node_id',     42,
          'class_filter_type', 'include',
          'class_filter_array', array( '13' ) ) )}
```

This example demonstrates how to use class filtering to fetch only nodes that reference objects that are instances of class number 13.

Example 7

```
{fetch( 'content', 'list',
    hash( 'parent_node_id',     42,
          'class_filter_type', 'include',
          'class_filter_array', array( 'folder' ) ) )}
```

This example demonstrates how to use class filtering to fetch only nodes that reference folder objects.

Example 8

```
{fetch( 'content', 'list',
    hash( 'parent_node_id',      42,
          'class_filter_type', 'exclude',
          'class_filter_array', array( 'article',
                                         'comment' ) ) )}
```

This example demonstrates how to use class filtering to exclude articles and comments from a fetch.

Example 9

```
{fetch( 'content', 'list',
    hash( 'parent_node_id',      42,
          'attribute_filter', array( 'or',
                                      array( 152, '=', 'abc' ),
                                      array( 153, '=', '42' ) ) ) )}
```

This example demonstrates how to do attribute filtering. Only nodes that have attributes number 152 and 153 set to "abc" and 42 respectively will be included in the result.

Example 10

```
{fetch( 'content',
        'list',
        hash( 'parent_node_id',      42,
              'attribute_filter', array( array( 'article/title',
                                              'like',
                                              '*story*' ) ) ) )}
```

This example demonstrates how to do attribute filtering. Instead of specifying the ID number of the attribute (as in the previous example), the identifier of the class and the attribute is used. Only article nodes that contain the text "story" in their title attribute will be included in the result.

Example 11

```
{fetch( 'content',
        'list',
        hash( 'parent_node_id',      42,
              'attribute_filter', array( array( 'article/image',
                                              '=',
                                              87 ) ) ) )}
```

This example demonstrates how to do attribute filtering using an attribute of the "Object relation" datatype. Only article nodes that store relation to the specified content object (object ID = 87) in their "image" attribute will be included in the result.

Example 12

```
{fetch( 'content',
       'list',
       hash( 'parent_node_id', 42,
             'attribute_filter', array( array( 'review/rating',
                                              'between',
                                              array( 0, 2 ) ) ) ) )}
```

This example demonstrates how to do attribute filtering using an attribute of the "Selection" datatype. Only review nodes that contain "Very good", "Good" or "Ok" (options 0, 1 or 2) in their "rating" attribute will be included in the result.

Example 13

```
{fetch( 'content',
       'list',
       hash( 'parent_node_id', 42,
             'attribute_filter', array( array( 'bug/priority',
                                              'in',
                                              array( 1, 4 ) ) ) ) )}
```

Fetches bugs with priority 1 and 4 (i.e. bug nodes that have their priority attribute set to 1 or 4).

Example 14

```
{fetch( 'content',
       'list',
       hash( 'parent_node_id', 42,
             'attribute_filter', array( array( 'bug/priority',
                                              'between',
                                              array( 2, 4 ) ) ) ) )}
```

Fetches bugs with priority 2, 3 and 4 (i.e. bug nodes that have their priority attribute value between 2 and 4).

Example 15

```
{fetch( 'content', 'list',
       hash( 'parent_node_id', 42,
             'attribute_filter', array( 'and',
                                         array( 'priority', '>', '20' ),
                                         array( 'article/
title', '=', 'abc' ) ) ) )}
```

This example demonstrates how to do filtering on a non-attribute. Instead of specifying the identifier of the class and the attribute, "priority" is used. Only article nodes that have their title attribute set to "abc" and have a priority greater than 20 will be included in the result.

Example 16

```
{def $nodes=fetch( 'content', 'list',
    hash( 'parent_node_id', 42,
        'attribute_filter', array(
            array( 'owner',
                '=',
                $current_user.contentobject_id ) ) ) }

{foreach $nodes as $node}
    {$node.name|wash}<br />
{/foreach}

{undef $nodes}
```

This example demonstrates how to do filtering on a non-attribute. Instead of specifying the identifier of the class and the attribute, "owner" is used. Only the nodes/objects that are created by the current user will be included in the result.

Example 17

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 2,
        'only_translated', true() ) )}
```

This example demonstrates how to fetch only nodes that reference objects that have been translated into the current language (the language that the siteaccess is using).

Example 18

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 2,
        'limit', 3,
        'only_translated', true(),
        'language', 'ger-DE' ) )}
```

This example demonstrates another way to fetch translated objects. The fetch will return nodes that reference objects which have been translated into the German language.

Example 19

```
{fetch( 'content', 'list',
    hash( 'parent_node_id', 2,
        'sort_by', array( 'priority', false() ),
        'limit', 15,
        'extended_attribute_filter',
            hash( 'id', 'MyFilter',
                'params', hash( 'value1', 'text1',
                    'value2', 'text2' ) ) ) )}
```

This example demonstrates how the extended attribute filter can be used. The template code above will fetch objects that match the following condition: any object using the ezstring datatype containing either "text1" or "text2". The necessary PHP implementation is shown in the example below.

```
class eZMyExtendedFilter
{
    /**
     * Constructor
     */
    function eZMyExtendedFilter()
    {
        // Empty...
    }

    function createSqlParts( $params )
    {
        $sqlTables= ', ezcontentobject_attribute as myfilter_alias ';

        $sqlJoins = ' ezcontentobject_tree.contentobject_id =
myfilter_alias.contentobject_id AND ezcontentobject_tree.contentobject_version
= myfilter_alias.version AND myfilter_alias.data_type_string = "ezstring"
AND';

        if ( isset( $params['value1'] ) )
        {
            $value1 = $params['value1'];
        }
        else
        {
            $value1 = 'fooo';
        }
        if ( isset( $params['value2'] ) )
        {
```

```
        $value2 = $params['value2'];
    }
    else
    {
        $value2 = 'boooo';
    }

    $sqlCondArray = array();

    $sqlCondArray[] = 'myfilter_alias.data_text = "' . $value1 . '"';
    $sqlCondArray[] = 'myfilter_alias.data_text = "' . $value2 . '"';

    $sqlCond = implode( ' or ', $sqlCondArray );

    $sqlCond = ' ( ' . $sqlCond . ' ) AND ' . $sqlJoins . ' ';

    return array( 'tables' => $sqlTables, 'joins' => $sqlCond );
}

}
```

Example 20

```
{fetch( 'content', 'list', hash( 'parent_node_id', 2,
                                'ignore_visibility', true() ) )}
```

This example demonstrates how to fetch nodes regardless of their visibility status. The operation above will return both visible and hidden nodes that are directly below node #2.

list_count

Summary

Fetches the number of children of a node.

Usage

```
fetch( 'content', 'list_count',
       hash( 'parent_node_id',      parent_node_id,
             [ 'class_filter_type',   class_filter_type,  ]
             [ 'class_filter_array',   class_filter_array, ]
             [ 'attribute_filter',    attribute_filter,   ]
             [ 'main_node_only',      main_node_only,     ] ) )
```

Parameters

Name	Type	Description	Required
parent_node_id	integer	The ID number of the parent node.	Yes.
class_filter_type	string	The type of class filtering (include/exclude).	No.
class_filter_array	array	The classes that should be filtered.	No.
attribute_filter	mixed	Filter logic for attribute level filtering.	No.
main_node_only	integer	Type of nodes that should be fetched (all or main nodes only).	No.

Returns

An integer (the number of nodes).

Description

This function operates in almost the same way as the "list" fetch function. The difference is that instead of returning the actual nodes, it returns the count (the number of nodes that were found). The "list_count" function takes the same parameters as the "list" function with some exceptions (sorting, grouping, limit/offset, etc. is not supported). Please refer to the documentation of the "list" (page [436](#)) function for a detailed description of the parameters.

Examples

Example 1

```
{def $count=fetch( 'content', 'list_count', hash( 'parent_node_id', 42 ) )}  
Node number 42 has {$count} number of children.
```

Outputs the number of nodes that are below node number 42.

locale_list

Summary

Fetches the available locales.

Usage

```
fetch( 'content', 'locale_list' )
```

Returns

An array of ezlocale (page 739) objects.

Description

This function fetches all the available locales and returns an array of ezlocale (page 739) objects.

Examples

Example 1

```
{def $locales=fetch( 'content', 'locale_list' )}

{foreach $locales as $locale}
    {$locale.locale_code} : {$locale.country_name} <br />
{/foreach}
```

Outputs the locale codes and the names of the countries that the different locales belong to.

navigation_part

Summary

Fetches information about a navigation part.

Usage

```
fetch( 'content', 'navigation_part', hash( 'identifier', identifier ) )
```

Parameters

Name	Type	Description	Required
identifier	string	The identifier of the desired navigation part.	Yes.

Returns

An associative array (see below) or FALSE.

Description

This function fetches information about a navigation part. The identifier of the navigation part must be specified using the "identifier" parameter. The function returns a hash that consists of the following elements:

Key	Type	Description
name	string	The actual name of the navigation part (for example "Content structure").
identifier	string	The identifier of the navigation part (for example "ezcontentnavigationpart").

Examples

Example 1

```
{def $navigation_part=fetch( 'content', 'navigation_part',
                           hash( 'identifier', 'ezcontentnavigationpart' ) )}
{$navigation_part.name}
```

Outputs the name of the navigation part identified by the string "ezcontentnavigationpart".

navigation_parts

Summary

Fetches all available navigation parts.

Usage

```
fetch( 'content', 'navigation_parts' )
```

Returns

An array of hashes (see below).

Description

This function fetches all the available navigation parts. The function returns an array with hashes containing the following elements:

Key	Type	Description
name	string	The actual name of the navigation part (for example "Content structure").
identifier	string	The identifier of the navigation part (for example "ez-contentnavigationpart").

Examples

Example 1

```
{def $navigation_parts=fetch( 'content', 'navigation_parts' )}

{foreach $navigation_parts as $navigation_part}
    {$navigation_part.identifier} : {$navigation_part.name} <br />
{/foreach}
```

Outputs the identifiers and names of all the available navigation parts.

node

Summary

Fetches a node (identified by either an ID number or a path).

Usage

```
fetch( 'content', 'node', hash( [ 'node_id', node_id, ]  
                                [ 'node_path', node_path ] ) )
```

Parameters

Name	Type	Description	Required
node_id	integer	The ID number of the node that should be fetched.	No.
node_path	string	The path of the node that should be fetched.	No.

Returns

An ezcontentobjecttreenode (page 721) object or FALSE.

Description

This function fetches a single node and returns it as a ezcontentobjecttreenode (page 721). The target node must be specified using either the "node_id" or the "node_path" parameter. If no node can be found, or if an error occurs, the function will return FALSE.

Examples

Example 1

```
{def $my_node=fetch( 'content', 'node', hash( 'node_id', 96 ) )}  
{$my_node.name|wash}
```

Fetches node number 96 and outputs the name of the object that is encapsulated by that node.

Example 2

```
{def $my_node=fetch( 'content', 'node', hash( 'node_path', 'news/
article_test' ) )}
{$my_node.name|wash}
```

Fetches the node by the specified path and outputs the name of the object that is encapsulated by that node.

non_translation_list

Summary

Fetches locales that a version of an object may be translated into.

Usage

```
fetch( 'content', 'non_translation_list', hash( 'object_id', id, 'version',
version )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
version	integer	The target version number.	Yes.

Returns

An array of ezlocale (page 453) objects or FALSE.

Description

This function will fetch all the locales that a specific version of a content object may be translated to. The locales which the version is already translated to will not be included. The function returns an array of ezlocale (page 453) objects. It will return FALSE if there are no more alternate locales.

Examples

Example 1

```
{def $locales=fetch( 'content', 'non_translation_list',
                    hash( 'object_id', 42,
                          'version',      3 ) )}

{foreach $locales as $locale}
    {$locale.language_name} <br />
{/foreach}
```

Outputs the language names of the locales that version number 3 of object number 42 can be translated to.

object

Summary

Fetches a content object (specified by an ID number).

Usage

```
fetch( 'content', 'object', hash( 'object_id', object_id ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.

Returns

An ezcontentobject (page 712) object or FALSE.

Description

This function fetches a content object. The ID number of the object must be specified using the "object_id" parameter. The function returns an ezcontentobject (page 712) object. It will return FALSE if a non-existing ID number is provided or if an error occurs.

Examples

Example 1

```
{def $object=fetch( 'content', 'object', hash( 'object_id', 13 ) )}  
{$object.name|wash}
```

Outputs the name of object number 13.

object_by_attribute**Summary**

DEPRECATED

object_count_by_user_id

Summary

Fetches the number of objects (of a class) created by a user.

Usage

```
fetch( 'content', 'object_count_by_user_id',
       hash( 'class_id', class_id,
             'user_id', user_id ) )
```

Parameters

Name	Type	Description	Required
class_id	integer	The ID number of the target class.	Yes.
user_id	integer	The ID number of the user (object ID).	Yes.

Returns

The number of objects (as an integer).

Description

This function counts the number of objects (of a certain type) that were created by a user. Both the type of the object (the class) and the user must be specified. The function returns an integer revealing the number of objects that were found.

Examples

```
{def count=fetch( 'content', 'object_count_by_user_id', hash( 'class_id', 13,
               'user_id', 14 ) )}
{$count}
{undef $count}
```

Outputs the number of objects (of class number 13) that have been created by user number 14.

pending_count

Summary

Fetches the number of pending objects for the current user.

Usage

```
fetch( 'content', 'pending_count' )
```

Returns

An integer revealing the number of pending objects.

Description

This function reveals the number of pending objects that belong to the current user.

Examples

Example 1

```
{def $count=fetch( 'content', 'pending_count' )}
There are {$count} pending objects.
```

Outputs the number of pending objects that belong to the current user.

pending_list

Summary

Fetches the pending objects for the current user.

Usage

```
fetch( 'content', 'pending_list' )
```

Returns

An array of (page 712) objects or FALSE.

Description

This function fetches the pending objects that belong to the current user. The function returns an array of (page 712) or FALSE if there are no pending objects.

Examples

Example 1

```
{def $pending=fetch( 'content', 'pending_list' )}

{foreach $pending as $object}
    {$object.name}
{/foreach}
```

Outputs the names of the pending objects that belong to the current user.

recent**Summary**

Fetches nodes where the current user recently published something.

Usage

```
fetch( 'content', 'recent' )
```

Returns

An array of ezcontentbrowserecent (page 704) objects or FALSE.

Description

This function fetches nodes under which the current user recently has published something. The function returns an array of ezcontentbrowserecent (page 704) objects. If there are no nodes under which the current user has published something, the function will return FALSE.

Examples**Example 1**

```
{def $recent=fetch( 'content', 'recent' )}
{foreach $recent as $element}
    {$element.name}
{/foreach}
```

Outputs the names of the nodes under which the current user recently published something.

related_objects

Summary

Fetches related objects.

Usage

```
fetch( 'content', 'related_objects',
      hash( 'object_id', object_id,
            [ 'attribute_identifier', attribute_identifier, ]
            [ 'all_relations', boolean, ]
            [ 'group_by_attribute', boolean, ],
            [ 'sort_by', sort_by ] ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
attribute_identifier	mixed	The ID number or class/attribute identifier of the target attribute.	No.
all_relations	boolean	Controls whether to fetch all types of relations or not, default is FALSE.	No.
group_by_attribute	boolean	Groups the result based on the attributes, default value is TRUE.	No.
sort_by	array	The sorting mechanism that should be used.	No.

Returns

An array of ezcontentobject (page 712) objects or a two-dimensional array if 'group_by_attribute' is TRUE. If no objects are found, the function will return FALSE.

Description

This function fetches the objects that have been related to an object specified by the "object_id" parameter. It returns all related objects regardless of their relation type (attributes using the "Object relation" (page 306) or "Object relations" (page 308) datatype or standard object-level relations).

The "attribute_identifier" parameter makes it possible to specify either an ID number or an identifier string (class/attribute - for example "my_class/my_attribute") of an attribute. This parameter is not required. The default value is zero, which makes the function return only objects that are related on an object level, not at the attribute level. This behavior is similar to 'related_

contentobject_array' functional attribute of a content object. When the parameter is used, the system will return objects that have been related using an attribute that is based on either the "Object relation" (page 306) or "Object relations" (page 308) datatype.

The "all_relations" parameter makes it possible to fetch all types of relations. This parameter is not required and the default value is FALSE.

The "group_by_attribute" parameter can only be used when the "all_relations" parameter has been set to TRUE. When the "group_by_attribute" parameter has been set to TRUE, the function will return a two-dimensional array instead of just an array of objects. The following example shows how this structure is built up:

```
$related_objects_grouped = array(  
    0 => array( $object1, $object2 ... ),  
        // Objects related on content object level  
        attr_id_1 => array( $object1, $object2 ... ),  
        attr_id_2 => array( $object1, $object2 ... ),  
        ...  
        // Objects related by attributes  
);
```

The "sort_by" parameter makes it possible to sort the result in different ways. This parameter must be provided as an array of arrays that define the sorting methods. The first element of each array must be the desired sorting method. The second element of the array must be the sorting direction, it can be either TRUE or FALSE (ascending or descending). Please note that this parameter works in the very same way as the "sort_by" parameter of the list (page 436) fetch function. However, it currently only supports the following sorting methods:

- class_identifier
- class_name
- modified
- name
- published
- section

Please note that using other sort methods will lead to an error.

Examples

Example 1

```
{def $related=fetch( 'content', 'related_objects',
hash( 'object_id', $node.object.id,
'all_relations', true(),
'group_by_attribute', true(),
'sort_by', array( array( 'class_name', true() ),
array( 'name', true() ) ) )}
```

Returns all relations grouped in arrays by attribute ID, then sorted by class name and by object's name in ascending order.

related_objects_count

Summary

Fetches the number of related objects.

Usage

```
fetch( 'content', 'related_objects_count',
      hash( 'object_id',                      object_id,
            [ 'attribute_identifier',           attribute_identifier,  ]
            [ 'all_relations',                 boolean,                ] ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
attribute_identifier	mixed	The ID number or class/attribute identifier of the target attribute.	No.
all_relations	boolean	Controls whether to fetch all types of relations or not, default is FALSE.	No.

Returns

An integer (the number of related objects).

Description

This fetch function operates in almost the same way as the "related objects" fetch function. The difference is that instead of returning the related objects themselves, it returns the count (the number of related objects that were found). The "related_objects_count" function takes the same parameters as the "related_objects" function with some exceptions (for example sorting is not supported). Please refer to the documentation of the "related_objects" (page 465) function for a detailed description of the parameters.

reverse_related_objects

Summary

Fetches reverse relates objects.

Usage

```
fetch( 'content', 'reverse_related_objects',
       hash( 'object_id', object_id,
             [ 'attribute_identifier', attribute_identifier, ]
             [ 'all_relations', boolean, ]
             [ 'group_by_attribute', boolean, ]
             [ 'sort_by', sort_by ] ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
attribute_identifier	mixed	The ID number or class/attribute identifier of the target attribute.	No.
all_relations	boolean	Controls whether to fetch all types of relations or not, default is FALSE.	No.
group_by_attribute	boolean	Groups the result based on the attributes, default value is TRUE.	No.
sort_by	array	The sorting mechanism that should be used.	No.

Returns

An array of ezcontentobject (page 712) objects or FALSE.

Description

This function makes it possible to fetch reverse related objects. The target object must be specified using the "object_id" parameter. The function will return an array of ezcontentobject (page 712) objects which are using the target object through the conventional object relation mechanism. If no objects are found, the function will return FALSE.

Class attribute filtering

By making use of the "attribute_identifier" parameter, it is possible to fetch reverse related objects that make use of the target object by the way of an attribute. The attribute must use either

the "Object relation" (page 306) or the "Object relations" (page 308) datatype. The "attribute_identifier" parameter can either be the ID number of the class attribute or a string that consists of the class identifier, a slash and the class attribute identifier (for example "my_class/my_attribute").

The "all_relations" parameter makes it possible to fetch all types of relations. This parameter is not required and the default value is FALSE.

The "group_by_attribute" parameter can only be used when the "all_relations" parameter has been set to TRUE. When the "group_by_attribute" parameter has been set to TRUE, the function will return a two-dimensional array instead of just an array of objects. The following example shows how this structure is built up:

```
$related_objects_grouped = array(  
 0 => array( $object1, $object2 ... ),  
    // Objects related on content object level  
    attr_id_1 => array( $object1, $object2 ... ),  
    attr_id_2 => array( $object1, $object2 ... ),  
    ...  
    // Objects related by attributes  
) ;
```

The "sort_by" parameter makes it possible to sort the result in different ways. This parameter must be provided as an array of arrays that define the sorting methods. The first element of each array must be the desired sorting method. The second element of the array must be the sorting direction, it can be either TRUE or FALSE (ascending or descending). Please note that this parameter works in the very same way as the "sort_by" parameter of the list (page 436) fetch function. However, it currently only supports the following sorting methods:

- class_identifier
- class_name
- modified
- name
- published
- section

Please note that using other sort methods will lead to an error.

Examples

Example 1

```
{def $objects=fetch( 'content', 'reverse_related_objects',
                    hash( 'object_id', 256 ) )}

{foreach $objects as $object}
    {$object.name|wash} <br />
{/foreach}
```

Outputs the names of the objects that make use of object number 256 through the conventional related objects mechanism.

Example 3

```
{def $objects=fetch( 'content', 'reverse_related_objects',
                    hash( 'object_id',           256,
                          'attribute_identifier', '4096' ) )

{foreach $objects as $object}
    {$object.name|wash} <br />
{/foreach}
```

Outputs the names of the objects that make use of object number 256 through class attribute number 4096.

Example 3

```
{def $objects=fetch( 'content', 'reverse_related_objects',
                    hash( 'object_id',           256,
                          'attribute_identifier', 'my_class/my_attribute' ) )

{foreach $objects as $object}
    {$object.name|wash} <br />
{/foreach}
```

Outputs the names of the objects that make use of object number 256 through an attribute called "my_attribute" that is a part of class "my_class".

reverse_related_objects_count

Summary

Fetches the number of reverse related objects.

Usage

```
fetch( 'content', 'reverse_related_objects_count',
       hash( 'object_id', object_id,
             [ 'attribute_identifier', attribute_identifier, ]
             [ 'all_relations', boolean, ] ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
attribute_identifier	mixed	The ID number or class/attribute identifier of the target attribute.	No.
all_relations	boolean	Controls whether to fetch all types of relations or not, default is FALSE.	No.

Returns

An integer (number of reverse related objects).

Description

This function is the same as the "reverse related objects" function. However, instead of returning an array of objects, it returns the number of objects that were found. Please refer to the documentation of the "reverse_related_objects" (page 469) function for information about the parameters.

same_classattribute_node

Summary

Fetches nodes containing attributes that match a certain value.

Usage

```
fetch( 'content', 'same_classattribute_node',
       hash( 'class_attribute_id', class_attribute_id,
             'value',           value,
             'datatype',        datatype ) )
```

Parameters

Name	Type	Description	Required
id	integer	The ID number of the class attribute that should be examined.	Yes.
value	mixed	The value that should be matched.	Yes.
datatype	string	Must be either "int", "float" or "text".	Yes.

Returns

An array of ezcontentobjecttreenode (page 721) objects or FALSE.

Description

This function will go through all object attributes that are instances of the class attribute specified by the "class_attribute_id" parameter. The value that should be matched must be specified using the "value" parameter. In addition, the type of data (either "int", "float" or "text") that the datatype representing the attribute must be provided. The function returns an array of ezcontentobjecttreenode (page 721) objects or FALSE (if there is no match).

Examples

```
{def $matched_nodes=fetch( 'content', 'same_classattribute_node',
                           hash( 'class_attribute_id', 245,
                                 'value',                 'example',
                                 'datatype',              'text' ) )}

{foreach $matched_nodes as $matched_node}
    {$matched_node.name|wash} <br />
{/foreach}
```

Outputs the names of the nodes that make use of class attribute number 245 and where the text contents of the object attribute equals the string "example".

search

Summary

Fetches nodes containing data that match a certain criteria.

Usage

```
fetch( content, search,
       hash( text, text,
              [ offset,                  offset, ]
              [ limit,                   limit, ]
              [ section_id,              id,      ]
              [ subtree_array,           array,   ]
              [ publish_timestamp,       time,    ]
              [ publish_date,            date,    ]
              [ class_id,                id,      ]
              [ class_attribute_id,      id,      ]
              [ sort_by,                 sort_by ] ) )
```

Parameters

Name	Type	Description	Required
text	string	The text that should be matched.	Yes.
subtree_array	mixed	Array node ID number under which the system should search.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of nodes that should be returned.	No.
publish_timestamp	integer	Only search objects with the specified publishing date/time (as a UNIX timestamp).	No.
publish_date	integer	Only search objects published during the last day / week / month / three months / one year.	No.
section_id	integer	Only match objects that are in this section.	No.
class_id	integer	Only match objects that are instances of this class. This parameter can also be an array of class ID numbers.	No.
class_attribute_id	integer	Only search within this attribute. This parameter can also be an array of attribute ID numbers.	No.
sort_by	mixed	Sort the result. See description below.	No.

Returns

An array of hashes (see below) or FALSE.

Description

This function will perform a search and it will return the hits that the current user has read access to. The function returns an array of hashes. The hashes consist of the following elements:

SearchResult	array	An array of the nodes (as ezcontentobjecttreenode (page 721) objects) that matched the search conditions.
SearchCount	integer	The total number of nodes that matched the search conditions.
StopWordArray	array	An array of strings containing words that were excluded from the search.

Sorting

The "sort_by" parameter makes it possible to sort the result in different ways. This parameter behaves exactly in the same way as it does for the "list" function. Please refer to the documentation page of the "list (page 436)" function for a complete explanation of this parameter.

Searching by the date of publishing

The "publish_date" parameter makes it possible to search objects published during the specified period. The following table reveals the possible values of this parameter.

1	one day
2	one week
3	one month
4	three months
5	one year

The "publish_timestamp" parameter makes it possible to search objects with the specified publishing date/time. This value must be a UNIX timestamp. You can also use an array of two elements in order to search within the given range.

Please note that you can not use both "publish_timestamp" and "publish_date" at the same time. If you use "publish_timestamp" then "publish_date" will be ignored.

Examples

Example 1

```
{def $search=fetch( 'content', 'search',
                    hash( 'text',      'example',
                          'class_id', array( '2', '5' ) ) )}
```

The search returned {\$search.SearchCount} matches.


```
{foreach $search.SearchResult as $matched_node}
    {$matched_node.name|wash} <br />
{/foreach}
```

Outputs the names of all nodes which encapsulte objects of classes 2 and 5 containing the word "example".

Example 2

```
{def $search=fetch( 'content', 'search',
                    hash( 'text',      'example',
                          'publish_timestamp', array( '1033920746',
                                         '1033920789' ) ) )}
```

The search returned {\$search.SearchCount} matches.


```
{foreach $search.SearchResult as $matched_node}
    {$matched_node.name|wash} <br />
{/foreach}
```

Outputs the names of all nodes which encapsulte objects published between the specified date/time values.

section_list

Summary

Fetches the available sections.

Usage

```
fetch( 'content', 'section_list' )
```

Returns

An array of ezsection (page 760) objects.

Description

This function fetches all the available sections. An array of ezsection (page 760) objects will be returned.

Examples

Example 1

```
{def $sections=fetch( 'content', 'section_list' )}
{foreach $sections as $section}
    {$section.name} <br />
{/foreach}
```

Outputs the names of all the available sections.

tipafriend_top_list

Summary

Fetches the most popular (most tipped) nodes.

Usage

```
fetch( 'content', 'tipafriend_top_list',
       hash( [ 'offset', offset,
               [ 'limit', limit ] ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	No.
limit	integer	The number of nodes that should be returned.	No.

Returns

An array of ezcontentobjecttreenode (page 721) objects or FALSE.

Description

This function fetches the nodes that were most tipped using the "Tip a friend" feature (for example "/content/tipafriend/44). The optional parameters "offset" and "limit" can be used to limit the result. The function returns an array of ezcontentobjecttreenode (page 721) objects. If no nodes are found or if an error occurs, the function will return FALSE.

Examples

Example 1

```
{def $popular_nodes=fetch( 'content', 'tipafriend_top_list', hash( 'limit',
10 ) )}

{foreach $popular_nodes as $popular_node}
    {$popular_node.name|wash} <br />
{/foreach}
```

Outputs the names of the ten most tipped nodes.

translation_list

Summary

Fetches the locales that can be used to translate objects.

Usage

```
fetch( 'content', 'translation_list' )
```

Returns

An array of ezlocale (page [739](#)) objects.

Description

This function fetches all the available locales that can be used to translate the contents of objects. This translations can be managed from within the "Translations" section of the "Setup" part within the administration interface. The function returns an array of ezlocale (page [739](#)) objects.

Examples

Example 1

```
{def $locales=fetch( 'content', 'translation_list' )}

{foreach $locales as $locale}
    {$locale.language_name} <br />
{/foreach}
```

Outputs the names of the languages that can be used to translate the contents of objects.

trash_count**Summary**

Fetches the number of objects that are in the trash.

Usage

```
fetch('content', 'trash_count' )
```

Returns

An integer revealing the number of objects that are in the trash.

Description

This function returns the number of objects (as an integer) that are in the trash. An object is considered to be in the trash if its status field is set to archived.

Examples**Example 1**

```
{def $trash_count=fetch('content', 'trash_count' )}  
There are {$trash_count} objects in the trash.
```

Outputs the number of items that are in the trash.

trash_object_list

Summary

Fetches the objects that are in the trash.

Usage

```
fetch( 'content', 'trash_object_list',
       hash( [ 'offset', offset,
               [ 'limit', limit ] ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	No.
limit	integer	The number of objects that should be returned.	No.

Returns

An array of ezcontentobject (page 712) objects or FALSE.

Description

This function fetches all the objects that are in the trash. An object is considered to be in the trash if its status field is set to "archived". The optional "offset" and "limit" parameters can be used to limit the result. The function returns an array of ezcontentobject (page 712) objects or FALSE if no objects are found.

Examples

```
{def $trashed_objects=fetch( 'content', 'trash_object_list' ) }

{foreach $trashed_objects as $object}
    {$object.name|wash} <br />
{/foreach}
```

Outputs the names of the objects that are in the trash.

tree

Summary

Fetches the children of a node recursively.

Usage

```
fetch( 'content', 'tree',
    hash( 'parent_node_id', parent_node_id,
        [ 'sort_by', sort_by, ] )
        [ 'offset', offset, ] )
        [ 'limit', limit, ] )
        [ 'attribute_filter', attribute_filter, ] )
        [ 'extended_attribute_filter', extended_attribute_filter, ] )
        [ 'class_filter_type', class_filter_type, ] )
        [ 'class_filter_array', class_filter_array, ] )
        [ 'only_translated', only_translated, ] )
        [ 'language', language, ] )
        [ 'main_node_only', main_node_only, ] )
        [ 'as_object', as_object, ] )
        [ 'depth', depth, ] )
        [ 'depth_operator', depth_operator, ] )
        [ 'limitation', limitation, ] )
        [ 'ignore_visibility', ignore_visibility, ] ) )
```

Parameters

Name	Type	Description	Required
parent_node_id	integer	The ID number of the parent node.	Yes.
sort_by	array	The sorting mechanism that should be used.	No.
offset	integer	The offset to start at.	No.
limit	integer	The maximum number of nodes that should be fetched.	No.
attribute_filter	mixed	Filter logic for attribute level filtering.	No.
class_filter_type	string	The type of class filtering (include/exclude).	No.
class_filter_array	array	The type of nodes that should be filtered.	No.
only_translated	boolean	Translation filtering (on/off).	No.
language	string	The language that should be filtered.	No.
main_node_only	boolean	Type of nodes that should be fetched (all or main nodes only).	No.
depth	integer	The maximum level of depth that should be explored.	No.
depth_operator	string	The logic to use when checking the depth.	No.
extended_attribute_filter	mixed	The extended attribute level filter logic.	No.
as_object	boolean	If TRUE (or omitted), an array of "ez-contentobjecttreenode" objects will be fetched. Otherwise, an array of arrays will be returned.	No.
limitation	array	Limitation array (empty array = access override).	No.
ignore_visibility	boolean	Makes it possible to fetch hidden nodes.	No.

Description

The "tree" function is very similar to the "list" fetch function. The only difference is that the tree function fetches child nodes recursively. The recursion depth can be controlled by the "depth" and "depth_operator" parameters. The rest of the parameters behave exactly in the same way as they do for the "list" function. Please refer to the documentation page of the "list (page 436)" function for a complete explanation of the parameters.

Depth

The depth parameter can be used to specify the level of depth (within the branch) that the function should explore when it is running. If the depth is set to one, this function will simply act

as the list function. If the depth is greater than one, the function will fetch nodes further down in the branch. The default value is unlimited.

Depth operator

The depth operator can be set to either "lt", "eq" or "gt" - meaning "less than", "equal to" and "greater than". For example, if it is set to 'eq', only nodes with the depth that was specified using the depth parameter will be fetched.

Examples

Example 1

```
{def $nodes=fetch( 'content', 'tree',
                  hash( 'parent_node_id', 42 ) )}

{foreach $nodes as $node}
    {$node.name|wash} <br />
{/foreach}
```

This example demonstrates how to fetch all the nodes that are under node number 42 recursively (all children, grand-children, etc. will be fetched). The names of the nodes are displayed.

tree_count

Summary

Fetches the number of children of a node recursively.

Usage

```
fetch( 'content', 'tree_count',
      hash( 'parent_node_id', parent_node_id,
            [ 'class_filter_type', class_filter_type, ]
            [ 'class_filter_array', class_filter_array, ]
            [ 'attribute_filter', attribute_filter, ]
            [ 'main_node_only', boolean, ]
            [ 'depth', depth, ]
            [ 'depth_operator', depth_operator ] ) )
```

Parameters

Name	Type	Description	Required
parent_node_id	integer	The ID number of the parent node.	Yes.
class_filter_type	string	Filter type for class filtering (include/exclude).	No.
class_filter_array	array	The type of nodes that should be filtered.	No.
attribute_filter	mixed	Filter logic for attribute level filtering.	No.
main_node_only	boolean	Type of nodes that should be fetched (all or main nodes only).	No.
depth	integer	The maximum level of depth that should be explored.	No.
depth_operator	string	The logic to use when checking the depth.	No.

Returns

An integer (number of nodes).

Description

This function works in the very same way as the "tree" function. The difference is that it returns only the number of nodes (instead of the actual nodes). Please refer to the documentation of the "list" (page 436), "list_count" (page 451) and the "tree" (page 483) fetch functions.

Examples

Example 1

```
{def $count=fetch( 'content', 'tree_count',
                  hash( 'parent_node_id',      42,
                        'class_filter_type', 'exclude',
                        'class_filter_array', array( 'folder', 'comment' ) )
                )}
Number of nodes: {$count}
```

This example counts the number of nodes that are children of node number 42, recursively. Nodes that reference "folder" or "comment" objects will be excluded from the count.

version**Summary**

Fetches a specific version of an object.

Usage

```
fetch( 'content', 'version', hash( 'object_id', object_id,
                                'version_id', version_id ) )
```

Parameters

Name	Type	Description	Required
object_id	integer	The ID number of the target object.	Yes.
version_id	integer	The version number that should be fetched.	Yes.

Returns

An ezcontentobjectversion (page 725) object or FALSE.

Description

This function fetches a specific version of a content object. Both the ID number of the target object and the number of the desired version must be specified using the "object_id" and the "version_id" parameters. The function returns an ezcontentobjectversion (page 725) object. It will return FALSE if invalid parameters have been provided or if the current user has insufficient permissions.

Examples**Example 1**

```
{def $version=fetch( 'content', 'version', hash( 'object_id', 13,
                                                'version_id', 3 ) )}
```

```
Name of version: {$version.name}
```

Outputs the name of version number 3 for object number 13.

version_count

Summary

Fetches the number of versions of a content object.

Usage

```
fetch( 'content', 'version_count', hash( 'contentobject', object ) )
```

Parameters

Name	Type	Description	Required
contentobject	object	The target object.	Yes.

Returns

The number of versions of an object (as an integer).

Description

This function retrieves the number of versions of a content object. The target object must be specified using the "contentobject" parameter.

Examples

Example 1

```
{* Fetch object number 13. *}
{def $object=fetch( 'content', 'object', hash( 'object_id', 13 ) )}

{* Fetch the number of versions for object number 13. *}
{def $versions=fetch( 'content', 'version_count', hash( 'contentobject',
$object ) )}

Object number 13 consists of {$versions} versions.
```

Outputs the number of versions that make up object number 13.

version_list

Summary

Fetches all the versions of a content object.

Usage

```
fetch( 'content', 'version_list', hash( 'contentobject', object,
                                      [ 'offset',          offset, ]
                                      [ 'limit',           limit ] ) )
```

Parameters

Name	Type	Description	Required
contentobject	object	The target object.	Yes.
offset	integer	Offset to start at.	No.
limit	integer	The number of versions that should be fetched.	No.

Returns

An array of ezcontentobjectversion (page 725) objects.

Description

This function fetches all the versions of a certain object. The object itself must be specified using the "contentobject" parameter. The "offset" and "limit" parameters are optional and can be used to limit the result. The function returns an array of ezcontentobjectversion (page 725) objects.

Examples

Example 1

```
{* Fetch object number 13. *}
{def $object=fetch( 'content', 'object', hash( 'object_id', 13 ) )}

{* Fetch all the versions of object number 13. *}
{def $versions=fetch( 'content', 'version_list', hash( 'contentobject',
$object ) )}

{* Loop through all versions and display their names. *}
{foreach $versions as $version}
```

```
{$version.name} <br />
{/foreach}
```

Outputs the names of all versions that belong to object number 13.

view_top_list

Summary

Fetches the most popular (most viewed) nodes.

Usage

```
fetch( 'content', 'view_top_list',
       hash( [ 'section_id', section_id, ]
             [ 'class_id',    class_id,   ]
             [ 'offset',      offset,     ]
             [ 'limit',       limit      ] ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the section.	No.
class_id	integer	The ID number of the class.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of nodes that should be returned.	No.

Returns

An array of ezcontentobjecttreenode (page 721) objects.

Description

This function fetches the most popular (most viewed) nodes. The function returns an array of ezcontentobjecttreenode (page 721) objects. The "section_id" and "class_id" parameters can be used to filter out objects of certain type and/or objects that belong to a certain section. The "offset" and "limit" parameters can be used to limit the result.

In order to work, this function requires the use of the cronjob script. A part of this script will update the view counters of the nodes by analyzing the Apache log files. The "Scripts[]" array of a configuration override for "cronjob.ini" should include the "updateviewcount.php" script:

```
...
Scripts []=updateviewcount.php
...
```

In addition, the logfile settings in "logfile.ini" should match the syntax of the Apache log files.

Examples

Example 1

```
{def $popular_nodes=fetch( 'content', 'view_top_list',
                           hash( 'class_id', 2,
                                 'limit', 10,
                                 'offset', 0 ) )}

{foreach $popular_nodes as $popular_node}
    {$popular_node.name|wash} <br />
{/foreach}
```

Outputs the names of the ten most popular nodes that encapsulate objects of class number 2.

Views

action (page 496)

Provides an interface to different actions (AddToBasket, SwapNode, etc.).

advancedsearch (page 497)

Provides the advanced search interface.

bookmark (page 498)

Provides an interface for managing the current user's bookmarks.

browse (page 499)

Provides an interface for selecting node(s) by browsing the node tree.

collectedinfo (page 500)

Provides an interface for displaying the information that was collected.

collectinformation (page 501)

Provides an interface for collecting information.

copy (page 502)

Provides an interface for copying a single node.

copysubtree (page 503)

Provides an interface for copying an entire subtree of nodes.

download (page 504)

Provides an interface for downloading files stored by the "File" datatype.

draft (page 505)

Provides an interface for managing the current user's drafts.

edit (page 506)

Provides an interface for editing and translating the contents of objects.

hide (page 507)

Provides an interface for hiding and revealing nodes.

keyword (page 508)

Loads a template that can fetch objects which use keyword.

move (page 509)

Provides an interface for changing the location of a node.

new (page 510)

Loads a template that can be used to display new content since last visit.

pdf (page 511)

Provides on-the-fly PDF generation of a node.

pendinglist (page 512)

Provides an overview of the current user's pending items.

removeassignment (page 513)

Provides an interface for removing node assignments.

removeeditversion (page 514)

Provides an interface for draft removal.

removenode (page 515)

Provides an interface for removing nodes.

removeobject (page 516)

Provides an interface for removing objects.

search (page 517)

Provides the standard search interface.

tipafriend (page 518)

Provides an interface to the "tip a friend" feature.

translate (page 519)

Provides an interface for the translation of a node (DEPRECATED).

translations (page 520)

Provides an interface for managing content translations.

trash (page 521)

Provides an interface for managing the trash.

upload (page 522)

Provides an interface for uploading a file which will become a node.

urltranslator (page 523)

Provides an interface for managing virtual URLs.

versions (page 524)

Provides an interface for managing the versions of an object.

versionview (page 525)

Provides an interface for viewing a version of an object.

view (page 526)

Provides an interface for viewing a node.

action**Summary**

Provides an interface to different actions (AddToBasket, SwapNode, etc.).

advancedsearch**Summary**

Provides the advanced search interface.

bookmark**Summary**

Provides an interface for managing the current user's bookmarks.

browse

Summary

Provides an interface for selecting node(s) by browsing the node tree.

collectedinfo**Summary**

Provides an interface for displaying the information that was collected.

collectinformation**Summary**

Provides an interface for collecting information.

copy

Summary

Provides an interface for copying a single node.

copySubtree**Summary**

Provides an interface for copying an entire subtree of nodes.

download**Summary**

Provides an interface for downloading files stored by the "File" datatype.

draft**Summary**

Provides an interface for managing the current user's drafts.

edit

Summary

Provides an interface for editing and translating the contents of objects.

hide

Summary

Provides an interface for hiding and revealing nodes.

keyword**Summary**

Loads a template that can fetch objects which use keyword.

move**Summary**

Provides an interface for changing the location of a node.

new

Summary

Loads a template that can be used to display new content since last visit.

pdf

Summary

Provides on-the-fly PDF generation of a node.

pendinglist**Summary**

Provides an overview of the current user's pending items.

removeassignment**Summary**

Provides an interface for removing node assignments.

removeeditversion**Summary**

Provides an interface for draft removal.

removenode**Summary**

Provides an interface for removing nodes.

removeobject**Summary**

Provides an interface for removing objects.

search

Summary

Provides the standard search interface.

tipafriend**Summary**

Provides an interface to the "tip a friend" feature.

translate**Summary**

Provides an interface for the translation of a node (DEPRECATED).

translations**Summary**

Provides an interface for managing content translations.

trash**Summary**

Provides an interface for managing the trash.

upload**Summary**

Provides an interface for uploading a file which will become a node.

urltranslator**Summary**

Provides an interface for managing virtual URLs.

versions**Summary**

Provides an interface for managing the versions of an object.

versionview**Summary**

Provides an interface for viewing a version of an object.

view

Summary

Provides an interface for viewing a node.

5.3.4 error

Summary

Provides an interface for error handling / reporting.

Description

This module is used internally by other modules that wish to report/display error messages. The view(s) that hte module provides can be used when developing/creating a custom module. It doesn't provide any direct/usable functionality when it comes to building a site (templatework only) with eZ publish.

5.3.5 ezinfo

Summary

Provides views for displaying information about eZ publish.

Description

This is a small module that makes it possible to extract some generic information about the system. It doesn't interface with any engines inside the kernel. What it does is that it provides three views: "about", "copyright" and "is_alive". The "about" view returns information about eZ publish, the "copyright" view returns copyright information (related to eZ publish) and the "is_alive" view checks the database connection and if everything seems to be okay, it returns the text "eZ publish is alive!".

The module components are documented in the following sections:

- Views (page [529](#))

Views

about (page 530)

Provides information about the system (version number, etc.).

copyright (page 531)

Provides copyright information related to eZ publish.

is_alive (page 532)

Provides information about the database connection.

about

Summary

Provides information about the system (version number, etc.).

copyright**Summary**

Provides copyright information related to eZ publish.

is_alive

Summary

Provides information about the database connection.

5.3.6 form

Summary

Provides a view that generates an E-mail containing the data that was posted.

Description

This module is one of the simplest modules in eZ publish. It only contains one view, which is the "process" view. This view can only be called using HTTP POST (form action). The process view simply sends out an e-mail containing the data that was posted using the input fields. This module does not act as an interface to an engine inside the eZ publish kernel. It only provides a simple form processing mechanism.

Please note that the view provided by this module is insecure by design. It is possible to specify the sender's and the receiver's E-mail address using certain hidden input fields. In other words, it can be easily exploited by spammers. That's why this module is disabled by default. It can be enabled by setting "Module=enabled" in the [FormProcessSettings] block within an configuration override file for "site.ini". Hint: by commenting out the sender/receiver lines within the switch statement in "/kernel/form/process.php" it is possible to make this module much more secure than it is by default.

The module components are documented in the following sections:

- Views (page [534](#))

Views

process (page 535)

Provides an interface for creating an E-mail based on form data.

process

Summary

Provides an interface for creating an E-mail based on form data.

5.3.7 infocollector

Summary

Provides views for managing collected information.

Description

This module provides various interfaces that can be used to inspect/view and delete information that was collected by content objects.

The module components are documented in the following sections:

- Views (page [537](#))

Views

collectionlist (page 538)

Provides an interface for viewing and removing the collections of an object.

overview (page 539)

Provides an interface for viewing objects that have collected information.

view (page 540)

Provides an interface for viewing and deleting a specific collection.

collectionlist**Summary**

Provides an interface for viewing and removing the collections of an object.

overview**Summary**

Provides an interface for viewing objects that have collected information.

view

Summary

Provides an interface for viewing and deleting a specific collection.

5.3.8 layout

Summary

Provides a view that makes it possible to use alternative pagelayouts.

Description

This module has only one view, "set". The view can be used to force the system to make use of a different pagelayout template than the default one (pagelayout.tpl). It simply takes care of setting the variable that eZ publish uses when picking out the pagelayout template file.

When finished, the module that is specified (after the layout part in the URL) will be executed. In other words, the layout module offers a prefix/prerun-mechanism that can be used to specify which pagelayout eZ publish should use when rendering a specific page. For example, it can be used to set the printer-friendly layout:

```
http://www.example.com/layout/set/print/content/view/full/45
```

Note that the URL actually contains two module-view pairs. The first module/view combination will make sure that the print pagelayout is used, the second module/view combination instructs eZ publish to display a full view of node number 45. The result will be the following: eZ publish will render the full view of node number 45, but instead of using "pagelayout.tpl" as the main template, it will use the pagelayout that is associated with "print" in the configuration override for "layout.ini".

The module components are documented in the following sections:

- Fetch functions (page [542](#))
- Views (page [544](#))

Fetch functions

sitedesign_list (page 543)

Fetches the names of the currently used designs.

sitedesign_list

Summary

Fetches the names of the currently used designs.

Usage

```
fetch( 'layout', 'sitedesign_list' )
```

Returns

An array of strings containing the design names.

Description

This function fetches the different designs that are used by the current siteaccess. The function returns an array of strings containing the names of the designs.

Examples

Example 1

```
{def $designs=fetch( 'layout', 'sitedesign_list' )}

{foreach $designs as $design}
    {$design} <br />
{/foreach}
```

Outputs the names of the designs that are used by the current siteaccess.

Views

set (page 545)

Forces the system to use an alternate pagelayout.

set

Summary

Forces the system to use an alternate pagelayout.

5.3.9 notification

Summary

Provides an interface to the notification engine.

Description

This module provides an interface to the notification engine inside the eZ publish kernel. The administration interface makes use of the views that this module provides in order to allow the management of notifications (add new, remove, edit, etc.). Although possible, it isn't common to use these views when building a website (unless there is a need to replicate the notification management functionality of the administration interface).

The module components are documented in the following sections:

- Fetch functions (page [547](#))
- Views (page [554](#))

Fetch functions

digest_handlers (page 548)

Fetches the notification handlers for the notification items that should be sent as digest to the current user.

digest_items (page 549)

Fetches notification items that should be sent as digest to the current user.

event_content (page 550)

Fetches the contents of the notification event.

handler_list (page 551)

Fetches the available notification handlers.

subscribed_nodes (page 552)

Fetches nodes that the current user has subscribed to.

subscribed_nodes_count (page 553)

Fetches the number of nodes that the current user has subscribed to.

digest_handlers**Summary**

Fetches the notification handlers for the notification items that should be sent as digest to the current user.

digest_items**Summary**

Fetches notification items that should be sent as digest to the current user.

event_content**Summary**

Fetches the contents of the notification event.

handler_list

Summary

Fetches the available notification handlers.

Usage

```
fetch( 'notification', 'handler_list' )
```

Returns

An array of notification handler objects.

Description

This function fetches all the available notification handlers and returns an array containing handler objects.

Examples

Example 1

```
{def $handlers=fetch( 'notification', 'handler_list' )}
{foreach $handlers as $handler}
    {$handler.id_string} <br />
{/foreach}
```

Outputs the identification string of all the available notification handlers.

subscribed_nodes

Summary

Fetches nodes that the current user has subscribed to.

Usage

```
fetch( 'notification', 'subscribed_nodes',
       hash( [ 'offset', offset,
               [ 'limit', limit ] ] ) )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	No.
limit	integer	The number of nodes that should be fetched.	No.

Returns

Array of ezsubtreenotificationrule (page 762) objects or FALSE.

Description

This function fetches the nodes that the current user has subscribed to. The "offset" and "limit" parameters are optional. The function returns an array of ezsubtreenotificationrule (page 762) objects or FALSE if no nodes could be found.

Examples

Example 1

```
{def $subscriptions=fetch( 'notification', 'subscribed_nodes' )}

{foreach $subscriptions as $subscription}
    {$subscription.node.name|wash} <br />
{/foreach}
```

Outputs the names of the nodes that the current user has subscribed to.

subscribed_nodes_count

Summary

Fetches the number of nodes that the current user has subscribed to.

Usage

```
fetch( 'notification', 'subscribed_nodes_count' )
```

Returns

The number of nodes that the current user has subscribed to (as an integer).

Description

This function fetches and returns the number of nodes that the current user has subscribed to.

Examples

Example 1

```
{def $node_count=fetch( 'notification', 'subscribed_node_count' )}  
The current user has subscribed to {$node_count} number of nodes.
```

Outputs the number of nodes that the current user has subscribed to.

Views

addtonotification (page 555)

Provides a mechanism that adds a new subtree notification.

runfilter (page 556)

Provides an interface for launching the main notification processing script and for generating a new time event.

settings (page 557)

Provides an interface for tweaking user notification settings.

addtonotification**Summary**

Provides a mechanism that adds a new subtree notification.

runfilter**Summary**

Provides an interface for launching the main notification processing script and for generating a new time event.

settings

Summary

Provides an interface for tweaking user notification settings.

5.3.10 package

Summary

Provides views for importing/exporting packages.

Description

This module provides an interface to the packaging system. It makes it possible to view, import, export, install, uninstall, etc. eZ publish packages. The views that this module provides are used within the setup part of the administration interface.

The module components are documented in the following sections:

- Fetch functions (page [559](#))
- Views (page [573](#))

Fetch functions

can_create (page 560)

Not documented yet.

can_edit (page 561)

Not documented yet.

can_export (page 562)

Not documented yet.

can_import (page 563)

Not documented yet.

can_install (page 564)

Not documented yet.

can_list (page 565)

Not documented yet.

can_read (page 566)

Not documented yet.

can_remove (page 567)

Not documented yet.

dependent_list (page 568)

Not documented yet.

item (page 569)

Not documented yet.

list (page 570)

Not documented yet.

maintainer_role_list (page 571)

Not documented yet.

repository_list (page 572)

Not documented yet.

can_create

Summary

Not documented yet.

can_edit

Summary

Not documented yet.

can_export

Summary

Not documented yet.

can_import

Summary

Not documented yet.

can_install

Summary

Not documented yet.

can_list

Summary

Not documented yet.

can_read

Summary

Not documented yet.

can_remove

Summary

Not documented yet.

dependent_list**Summary**

Not documented yet.

item

Summary

Not documented yet.

list

Summary

Not documented yet.

maintainer_role_list**Summary**

Not documented yet.

repository_list**Summary**

Not documented yet.

Views

create (page 574)

Not documented yet.

export (page 575)

Not documented yet.

install (page 576)

Not documented yet.

list (page 577)

Not documented yet.

uninstall (page 578)

Not documented yet.

upload (page 579)

Not documented yet.

view (page 580)

Not documented yet.

create

Summary

Not documented yet.

export

Summary

Not documented yet.

install

Summary

Not documented yet.

list

Summary

Not documented yet.

uninstall**Summary**

Not documented yet.

upload

Summary

Not documented yet.

view

Summary

Not documented yet.

5.3.11 pdf

Summary

Provides views for configuring PDF exports.

Description

This module provides an interface to the PDF (Portable Document Format) engine inside the kernel. This module has a couple of views that make it possible to configure PDF exports. The views provided by this module are used within the setup part of the administration interface.

The module components are documented in the following sections:

- Views (page [582](#))

Views

edit (page 583)

Provides an interface for editing PDF exports.

list (page 584)

Provides an interface for generating an overview of the PDF exports.

edit

Summary

Provides an interface for editing PDF exports.

list

Summary

Provides an interface for generating an overview of the PDF exports.

5.3.12 **reference**

Summary

Provides a view for displaying documentation generated by Doxygen.

Description

This module provides a view for displaying documentation generated by Doxygen.

5.3.13 role

Summary

Provides views for managing roles.

Description

This module provides an interface to the permission system inside the eZ Publish kernel. It contains views that can be used to manage roles, role assignments, policies and so on. These views are used by the administration interface.

The module components are documented in the following sections:

- Views (page [587](#))

Views

assign (page 593)

Provides an interface for assigning roles to users and user groups.

copy (page 592)

Provides an interface for copying roles.

edit (page 591)

Provides an interface for editing roles.

list (page 590)

Provides an interface for generating an overview of all available roles.

policyedit (page 589)

Provides an interface for editing policies.

view (page 588)

Provides an interface for viewing a role.

view

Summary

Provides an interface for viewing a role.

policyedit**Summary**

Provides an interface for editing policies.

list

Summary

Provides an interface for generating an overview of all available roles.

edit

Summary

Provides an interface for editing roles.

copy

Summary

Provides an interface for copying roles.

assign

Summary

Provides an interface for assigning roles to users and user groups.

5.3.14 rss

Summary

Provides views for managing RSS imports and exports.

Description

This module provides an interface to the RSS (Really Simple Syndication) engine inside the eZ Publish kernel. The views that this module provide are used by the administration interface. These views make it possible to view and mange incoming and outgoing RSS feeds.

The module components are documented in the following sections:

- Views (page [595](#))

Views

edit_export (page 596)

Provides an interface for editing an RSS export.

edit_import (page 597)

Provides an interface for editing an RSS import.

feed (page 598)

Not documented yet.

list (page 599)

Provides an interface for generating an overview of RSS imports and exports.

edit_export**Summary**

Provides an interface for editing an RSS export.

edit_import**Summary**

Provides an interface for editing an RSS import.

feed

Summary

Not documented yet.

list

Summary

Provides an interface for generating an overview of RSS imports and exports.

5.3.15 search

Summary

Provides a view that displays search statistics.

Description

This module provides only a single view which is the "stats" view. The view can be used to display and reset the search statistics. The search module itself doesn't take care of indexing words, etc. This is done from within the content engine when objects are published.

The module components are documented in the following sections:

- Views (page [601](#))

Views

stats (page 602)

Provides an interface for viewing and resetting the search statistics.

stats

Summary

Provides an interface for viewing and resetting the search statistics.

5.3.16 section

Summary

Provides views for managing sections.

Description

This module provides an interface to the content engine inside the eZ Publish kernel. The section mechanism can be used to create groups of objects. Please refer to the "sections" (page 136) part of the "Concepts and basics" chapter for more information about sections. This module provides views that are used by the administration interface.

The module components are documented in the following sections:

- Fetch functions (page 604)
- Views (page 612)

Fetch functions

object (page 605)

Fetches a section.

object_list (page 606)

Fetches objects that belong to certain section.

object_list_count (page 608)

Fetches the number of objects that belong to certain section.

roles (page 609)

Fetches roles that have at least one policy limited to a certain section.

user_roles (page 611)

Fetches users and/or user groups with role limitations related to a certain section.

object

Summary

Fetches a section.

Usage

```
fetch( 'section', 'object', hash( 'section_id', section_id ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the section that should be fetched.	Yes.

Returns

An ezsection (page 760) object or FALSE.

Description

This function fetches an object that represents a section. The "section_id" parameter must be a valid section ID number. The function returns an ezsection (page 760) object. If an invalid ID number is specified, the function will return FALSE.

Examples

Example 1

```
{def $section=fetch( 'section', 'object', hash( 'section_id', 13 ) )}  
{$section.name|wash}
```

Outputs the name of section number 13.

object_list

Summary

Fetches objects that belong to certain section.

Usage

```
fetch( 'section', 'object_list',
       hash( 'section_id', section_id,
             [ 'offset',      offset,      ]
             [ 'limit',       limit,       ]
             [ 'sort_order', sort_order ] ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the target section.	Yes.
offset	integer	The offset to start at.	No.
limit	integer	The number of objects that should be fetched.	No.
sort_order	array	The desired sorting order.	No.

Returns

An array of ezcontentobject (page 712) objects or FALSE.

Description

This function fetches a set of objects that belong to a certain section. The section must be specified by its ID number using the "section_id" parameter. The "offset", "limit" and "sort_order" parameters work in the same way as in the list (page 436) fetch function of the "content" module. The function returns an array of ezcontentobject (page 712) objects. If no objects can be found, or if the provided section ID number is invalid, FALSE will be returned.

Examples

Example 1

```
{def $objects=fetch( 'section', 'object_list',
                     hash( 'section_id', 13 ) )}

{foreach $objects as $object}
```

```
{$object.name} <br />
{/foreach}
```

Outputs the names of the objects that belong to section number 13.

object_list_count

Summary

Fetches the number of objects that belong to certain section.

Usage

```
fetch( 'section', 'object_list_count', hash( 'section_id', section_id ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the target section.	Yes.

Returns

The number of objects (as an integer) that belong to the section.

Description

This function counts the number of objects that belong to a section specified by the "section_id" parameter. The function returns the count as an integer.

Examples

Example 1

```
{def $count=fetch( 'section', 'object_list_count',
                  hash( 'section_id', 13 ) )}
```

```
There are {$count} number of objects in section number 13.
```

Outputs the number of objects that belong to section number 13.

roles

Summary

Fetches roles that have at least one policy limited to a certain section.

Usage

```
fetch( 'section', 'roles', hash( 'section_id', section_id ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the target section.	Yes.

Returns

An array (see below) or FALSE.

Description

This function returns a structure that contains information about roles which have at least one policy limited to a certain section. The function returns an array with two keys:

Name	Description
roles	Contains a list of roles with at least one policy limited to the given section.
limited_policies	Contains a 2D array (the role ID as the first key) of the limited policies.

Examples

Example 1

```
{def $roles_array=fetch( 'section', 'roles', hash( 'section_id', 13 ) )
$roles=$roles_array.roles
$policies=$roles_array.limited_policies}

{foreach $roles as $role}
{$role.name}:
{foreach $limited_policies[$role.id] as $policy}
{$policy.module_name}/{$policy.function_name}
{delimiter}, {/delimiter}
```

```
{/foreach}  
<br />  
{/foreach}
```

Outputs information about roles that have limitations associated with section number 13.

user_roles

Summary

Fetches users and/or user groups with role limitations related to a certain section.

Usage

```
fetch( 'section', 'user_roles', hash( 'section_id', section_id ) )
```

Parameters

Name	Type	Description	Required
section_id	integer	The ID number of the target section.	Yes.

Returns

An array (see below) or FALSE.

Description

This function fetches user and/or user groups that have role limitations associated with a certain section. The section must be defined using a valid section ID number through the "section_id" parameter. Please refer to the example below to see how the returned information can be used.

Examples

Example 1

```
{def $user_roles=fetch( 'section', 'user_roles', hash( 'section_id', 13 ) )}

{foreach $user_roles as $user_role}
    User (or group) {$user_role.user.name} has limitation on the section 13.
    The "touched" role is {$user_role.role.name}. <br />
{/foreach}
```

Views

assign (page 616)

Provides an interface for assigning objects to a section.

edit (page 615)

Provides an interface for editing a section.

list (page 614)

Provides an interface for generating an overview of all the available sections.

view (page 613)

Provides an interface for viewing a section.

view

Summary

Provides an interface for viewing a section.

list

Summary

Provides an interface for generating an overview of all the available sections.

edit

Summary

Provides an interface for editing a section.

assign

Summary

Provides an interface for assigning objects to a section.

5.3.17 setup

Summary

Provides the web based setup wizard.

Description

This module provides the views for the web based setup wizard.

5.3.18 shop

Summary

Provides views for the webshop (basket, wish list, order list, etc.).

Description

This module provides an interface to the content and the e-commerce engine inside the eZ Publish kernel. The views that this module offers are typically used when building a webshop oriented site. This module also provides some views that are used by the administration interface.

The module components are documented in the following sections:

- Fetch functions (page [619](#))
- Views (page [628](#))

Fetch functions

basket (page 620)

Fetches the current user's shopping basket.

best_sell_list (page 621)

Fetches the most popular / most sold products.

related_purchase (page 623)

Fetches products that were purchased together with a given product.

wish_list (page 625)

Fetches the products of a given wishlist.

wish_list_count (page 627)

Fetches a wishlist and returns the number of items in it.

basket**Summary**

Fetches the current user's shopping basket.

Usage

```
fetch( 'shop', 'basket' )
```

Returns

An ezbasket (page 700) object.

Description

This function fetches the shopping basket that belongs to the current user. The function returns an ezbasket (page 700) object.

Examples**Example 1**

```
{def $basket=fetch( 'shop', 'basket' )}

{if $basket.is_empty}

    There are no products in the basket.

{else}

    There are {count( $basket.items )} items in the basket. <br />
    Total price (ex. VAT) : {$basket.total_ex_vat|l10n( currency )} <br />
    Total price (inc. VAT): {$basket.total_inc_vat|l10n( currency )}

{/if}
```

Outputs basic information about the current user's shopping basket.

best_sell_list

Summary

Fetches the most popular / most sold products.

Usage

```
fetch( 'shop', 'best_sell_list',
       hash( 'top_parent_node_id', id,
             'limit',           limit ) )
```

Parameters

Name	Type	Description	Required
top_parent_node_id	integer	The ID number of the top node.	Yes.
limit	integer	The number of objects that should be returned.	Yes.

Returns

An array of ezcontentobject (page 712) objects or FALSE.

Description

This function fetches the most popular / most sold products that are located within a part of the content node tree. The "top_parent_node_id" parameter must be used to tell the function under which node it should look for popular products. The "limit" parameter controls the number of items that will be returned. The function returns an array of ezcontentobject (page 712) objects. If the function is unable to find any products, FALSE will be returned.

Examples

Example 1

```
{def $best_sellers=fetch( 'shop', 'best_sell_list',
                         hash( 'top_parent_node_id', 2,
                               'limit',           5 ) )}

{foreach $best_sellers as $product}
    {$product.name} <br />
{/foreach}
```

Outputs the names of the five most popular products that are located somewhere below node number 2.

related_purchase

Summary

Fetches products that were purchased together with a given product.

Usage

```
fetch( 'shop', 'related_purchase',
       hash( 'contentobject_id', id,
             'limit',           limit ) )
```

Parameters

Name	Type	Description	Required
contentobject_id	integer	The ID number of the object representing the source product.	Yes.
limit	integer	The number of objects that should be fetched.	Yes.

Returns

An array of ezcontentobject (page 712) objects or FALSE.

Description

This function fetches a collection of products (content objects) that were purchased together with a given product. It can be used to create a "People who bought this product has also bought..." list. The source product must be specified using the "contentobject_id" parameter. This parameter must be an integer that reveals the ID number of the content object that represents the source product. The "limit" parameter must be used to limit the result. The function will return an array of ezcontentobject (page 712) objects or FALSE if no objects were found.

Examples

Example 1

```
{def $other_products=fetch( 'shop', 'related_purchase',
                           hash( 'contentobject_id', 32,
                                 'limit',           5 ) )}

{foreach $other_products as $product}
  {$product.name} <br />
```

```
{/foreach}
```

Outputs the names of 5 products that were bought together with a product represented by object number 32.

wish_list

Summary

Fetches the products of a given wishlist.

Usage

```
fetch( 'shop', 'wish_list',
       hash( 'production_id', production_id,
             [ 'offset', offset, ]
             [ 'limit', limit ] ) )
```

Parameters

Name	Type	Description	Required
production_id	integer	The ID of the target wishlist.	Yes.
offset	integer	The offset to start at.	No.
limit	integer	The number of products that should be fetched.	No.

Returns

An array of arrays containing information about the items in the wishlist (see below) or FALSE.

Description

This function fetches the products that belong to a certain wishlist. The system stores wishlists using the same solution that is used to store the contents of shopping baskets, the "product collection" schema. The desired wishlist must be specified using the ID number of the product collection that contains the wishlist. The optional "offset" and "limit" parameters can be used to control the resulting set. The function returns an array of arrays containing information about each item in the wishlist. The following table shows the structure that is used for each element of the array.

Attribute	Type	Description
id	string	The ID number of the item in the basket.
vat_value	string	32
item_count	string	The quantity.
node_id	string	The ID number of the node that represents the item.
object_name	string	The name of the object that

		represents the item.
price_ex_vat	double	The price of the item excluding the VAT.
price_inc_vat	string	The price of the item including the VAT.
discount_percent	string	The discount percentage (if any).
total_price_ex_vat	double	The total price excluding the VAT.
total_price_inc_vat	double	The total price including the VAT.
item_object	object	The item itself (as an ezproductcollectionitem (page 757) object).

If the system is unable to find any products, an empty wishlist / FALSE will be returned.

Examples

Example 1

```
{def $wishlist=fetch( 'shop', 'wish_list', hash( 'production_id', 13 ) )}

{foreach $wishlist as $wish}
    {$wish.object_name|wash} <br />
{/foreach}
```

Outputs the names of the objects that make up wishlist number 13.

wish_list_count

Summary

Fetches a wishlist and returns the number of items in it.

Usage

```
fetch( 'shop', 'wish_list_count', hash( 'production_id', production_id ) )
```

Parameters

Name	Type	Description	Required
production_id	integer	The ID number of the target wishlist.	Yes.

Returns

The number of items that make up the wishlist (as an integer).

Description

This function fetches the products that belong to a certain wishlist and returns an integer. The integer reveals the number of items that make up the wishlist. The system stores wishlists using the same solution that is used to store the contents of shopping baskets, the "product collection" schema. The desired wishlist must be specified using the ID number of the product collection that contains the target wishlist.

Examples

Example 1

```
{def $sum=fetch( 'shop', 'wish_list_count', hash( 'productcollection_id', 13 ) )}
```

```
There are {$sum} items in wishlist number 13.
```

Outputs the number of items that make up wishlist number 13.

Views

basket (page 629)

Provides an interface to the shopping basket of the current user.

checkout (page 630)

Provides the checkout interface.

confirmorder (page 631)

Provides the interface that asks the user to confirm an order.

customerlist (page 632)

Provides an interface for generating an overview of the customers.

customerorderview (page 633)

Provides an interface for viewing information the orders of a customer.

discountgroup (page 634)

Provides an interface for generating an overview of the discount groups.

discountgroupedit (page 635)

Provides an interface for editing a discount group.

discountgroupview (page 636)

Provides an interface for viewing a discount group.

discountruleedit (page 637)

Provides an interface for editing a discount rule.

orderlist (page 638)

Provides an interface for generating an overview of orders.

orderview (page 639)

Provides an interface for viewing an order.

register (page 640)

Provides an interface for registering a customer.

removeorder (page 641)

Provides an interface for removing an order.

statistics (page 642)

Provides an interface for generating sales statistics.

userregister (page 643)

Provides an interface for registering a user.

vattype (page 644)

Provides an interface for managing VATs.

wishlist (page 645)

Provides an interface for viewing and managing the current user's wishlist.

basket**Summary**

Provides an interface to the shopping basket of the current user.

checkout**Summary**

Provides the checkout interface.

confirmorder**Summary**

Provides the interface that asks the user to confirm an order.

customerlist**Summary**

Provides an interface for generating an overview of the customers.

customerorderview**Summary**

Provides an interface for viewing information the orders of a customer.

discountgroup**Summary**

Provides an interface for generating an overview of the discount groups.

discountgroupedit**Summary**

Provides an interface for editing a discount group.

discountgroupview**Summary**

Provides an interface for viewing a discount group.

discountruleedit**Summary**

Provides an interface for editing a discount rule.

orderlist**Summary**

Provides an interface for generating an overview of orders.

orderview**Summary**

Provides an interface for viewing an order.

register

Summary

Provides an interface for registering a customer.

removeorder**Summary**

Provides an interface for removing an order.

statistics**Summary**

Provides an interface for generating sales statistics.

userregister**Summary**

Provides an interface for registering a user.

vattype

Summary

Provides an interface for managing VATs.

wishlist**Summary**

Provides an interface for viewing and managing the current user's wishlist.

5.3.19 trigger

Summary

Provides a view for managing workflow triggers.

Description

This module provides an interface to the workflow engine inside the eZ Publish kernel. It consists of a view that can be used to list and manage the workflows that should be triggered before or after a specific function within a specific module is executed. The administration interface makes use of this view to allow the administrator to view and manage triggers.

The module components are documented in the following sections:

- Views (page [647](#))

Views

list (page 648)

Provides an interface for viewing and managing the workflow triggers.

list

Summary

Provides an interface for viewing and managing the workflow triggers.

5.3.20 url

Summary

Provides views for managing the URLs stored in the database.

Description

This module provides an interface to the content engine inside the eZ Publish kernel. Every address that is input as a link into an attribute using the XML block (page 329) or the URL (page 325) datatype is stored in a separate part of the database. Actual data stored using these datatypes only contain references to entries in the separate URL table. This feature makes it possible to inspect and edit the published URLs without having to interact with the content objects. Please refer to the "URL storage" (page 138) part of the "Concepts and basics" chapter for more information about this feature. The "url" module provides views that make it possible to manage the URLs. The administration interface makes use of this module to allow the users to manage the URLs.

The module components are documented in the following sections:

- Fetch functions (page 650)
- Views (page 654)

Fetch functions

list (page 651)

Fetches the URLs that are stored in the URL table.

list_count (page 653)

Fetches the number of URLs that are stored in the URL table.

list

Summary

Fetches the URLs that are stored in the URL table.

Usage

```
fetch( 'url', 'list', hash( [ 'is_valid', is_valid, ]
                           [ 'offset',    offset,    ]
                           [ 'limit',     limit      ] ) )
```

Parameters

Name	Type	Description	Required
is_valid	boolean	Instructs the system to only fetch valid or invalid URLs.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of URLs that should be fetched.	No.

Returns

An array of ezurl (page 764) objects or FALSE.

Description

This function fetches URLs from the URL table. The URL table stores addresses that have been input using the URL (page 325) or the XML block (page 329) datatype. Please refer to the "URL storage" (page 138) section of the "Concepts and basics" chapter for more information about how the system handles URLs.

The "is_valid", "offset" and "limit" parameters are optional. While the "is_valid" parameter can be used to filter out only valid (TRUE) or invalid (FALSE) URLs, the "offset" and "limit" parameters can narrow down the result. If the "is_valid" parameter is omitted, both valid and invalid URLs will be fetched. The function returns an array of ezurl (page 764) objects. If no URLs can be found, the function will return FALSE.

Examples

Example 1

```
{def $urls=fetch( 'url', 'list', hash( 'is_valid', true(),
                                      'offset', 0,
                                      'limit', 10 ) )}

{foreach $urls as $url}
    {$url.url} <br />
{/foreach}
```

Outputs the first ten valid URLs that are stored in the URL table.

list_count

Summary

Fetches the number of URLs that are stored in the URL table.

Usage

```
fetch( 'url', 'list_count', hash( [ 'is_valid', is_valid ] ) )
```

Parameters

Name	Type	Description	Required
is_valid	boolean	Instructs the system to count either valid or invalid URLs.	No.

Returns

The number of URLs (as an integer).

Description

This function fetches and counts the URLs that are stored in the URL table. The URL table stores addresses that have been input using the URL (page 325) or the XML block (page 329) datatype. Please refer to the "URL storage" (page 138) section of the "Concepts and basics" chapter for more information about how the system handles URLs.

The "is_valid" parameter is optional and can be used to filter out only valid (TRUE) or invalid (FALSE) URLs. If the "is_valid" parameter is omitted, both valid and invalid URLs will be counted. The function returns the number of found URLs as an integer.

Examples

Example 1

```
{def $valid_urls=fetch( 'url', 'list_count', hash( 'is_valid', true() ) )
Number of valid URLs: {$valid_urls}}
```

Outputs the number of valid URLs.

Views

edit (page 657)

Provides an interface for editing a URL.

list (page 656)

Provides an interface for generating an overview of all URLs.

views (page 655)

Provides an interface for viewing an URL.

views**Summary**

Provides an interface for viewing an URL.

list

Summary

Provides an interface for generating an overview of all URLs.

edit

Summary

Provides an interface for editing a URL.

5.3.21 user

Summary

Provides views for logging users in/out, password changing, etc.

Description

This module provides an interface to the permission system inside the eZ Publish kernel. It contains views that make it possible to log users in and out, register and activate new users, password changing, etc. A typical eZ Publish site that has login capabilities makes use of the views that this module provides.

The module components are documented in the following sections:

- Fetch functions (page [659](#))
- Views (page [673](#))

Fetch functions

anonymous_count (page 660)

Fetches the number of anonymous users.

current_user (page 661)

Fetches the user that is currently logged in.

has_access_to (page 662)

Checks if a user has access to a certain function of a module.

is_logged_in (page 664)

Checks if a specific user is logged in.

logged_in_count (page 665)

Fetches the number of users that are logged in.

logged_in_list (page 666)

Fetches the names of the users that are logged in.

logged_in_users (page 668)

Fetches the users that are logged in.

member_of (page 670)

Fetches the roles that are assigned to a user.

user_role (page 671)

Fetches the policies that are available for a user.

anonymous_count

Summary

Fetches the number of anonymous users.

Usage

```
fetch( 'user', 'anonymous_count' )
```

Returns

The number of anonymous users as an integer.

Description

This function counts the number of anonymous users currently accessing the site and returns that count (as an integer). An anonymous user is considered to be active if the last access time within the range of the activity timeout. The timeout can be set using the "ActivityTimeout" directive in a configuration override for "site.ini". The default timeout is one hour.

Examples

Example 1

```
{def $visitors=fetch( 'user', 'anonymous_count' )}  
There are {$visitors} anonymous users accessing the site.
```

Outputs the number of anonymous users that are currently accessing the site.

current_user

Summary

Fetches the user that is currently logged in.

Usage

```
fetch( 'user', 'current_user' )
```

Returns

An ezuser (page [765](#)) object.

Description

This function fetches the user object for the user that is currently logged in. If no user is logged in, the anonymous user will be returned. In both cases, the function will return an ezuser (page [765](#)) object.

Examples

Example 1

```
{def $user=fetch( 'user', 'current_user' )}
User:    {$user.contentobject.name} <br />
E-mail:  {$user.email} <br />
Username: {$user.login} <br />
Group(s): {$user.groups|implode( ', ')} <br />
```

Outputs miscellaneous information about the user that is currently logged in.

has_access_to

Summary

Checks if a user has access to a certain function of a module.

Usage

```
fetch( 'user', 'has_access_to',
       hash( 'module', module
             'function', function,
             [ 'user_id', user_id ] ) )
```

Parameters

Name	Type	Description	Required
module	string	The name of the module.	Yes.
function	string	The name of the function.	Yes.
user_id	integer	The ID number of the user.	No.

Returns

TRUE if access is allowed, FALSE otherwise.

Description

This function checks if the current user has access to a certain function of a module. The name of the module and the function must be provided using the "module" and the "function" parameters. The optional "user id" parameter can be used to check access for other users than the current user. The function returns TRUE if access is allowed, otherwise FALSE will be returned.

Examples

Example 1

```
{def $access=fetch( 'user', 'has_access_to',
                    hash( 'module', 'content',
                          'function', 'read',
                          'user_id', 128 ) )}

{if $access}
    Access is allowed.
{else}
```

```
Access is denied.  
{/if}
```

Reveals if user number 128 has access to the read function of the content module.

is_logged_in

Summary

Checks if a specific user is logged in.

Usage

```
fetch( 'user', 'is_logged_in', hash( 'user_id', user_id ) )
```

Parameters

Name	Type	Description	Required
user_id	integer	The ID number of the user that should be checked.	Yes.

Returns

TRUE if the specified user is logged in, FALSE otherwise.

Description

This function checks if a user is logged in or not. The desired user's ID number must be specified using the "user_id" parameter. The ID number of a user is the same as the ID number of the content object that represents that user. A user is considered to be active / logged in if the last access time is within the range of the activity timeout. The timeout can be set using the "ActivityTimeout" directive in a configuration override for "site.ini". The default timeout is one hour.

Examples

Example 1

```
{def $test=fetch( 'user', 'is_logged_in', hash( 'user_id', 256 ) )}

{if $test}
    User number 256 is currently logged in.
{else}
    User number 256 is not logged in.
{/if}
```

Outputs information that reveals whether user number 256 is logged in or not.

logged_in_count

Summary

Fetches the number of users that are logged in.

Usage

```
fetch( 'user', 'logged_in_count' )
```

Returns

The number of logged in users (as an integer).

Description

This function counts the number of logged in users (both anonymous and non-anonymous) and returns that count as an integer. A user is considered to be active / logged in if the last access time is within the range of the activity timeout. The timeout can be set using the "ActivityTimeout" directive in a configuration override for "site.ini". The default timeout is one hour.

Examples

Example 1

```
{def $users=fetch( 'user', 'logged_in_count' )}  
There are currently {$users} active users on the system.
```

Outputs the number of currently active / logged in users.

logged_in_list

Summary

Fetches the names of the users that are logged in.

Usage

```
fetch( 'user', 'logged_in_list',
       hash( [ 'sort_by', sort_by,
               [ 'offset', offset,
                 [ 'limit', limit ] ] ] ) )
```

Parameters

Name	Type	Description	Required
sort_by	mixed	The field that should be used by the sorting mechanism.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of users that should be fetched.	No.

Returns

An associative array or FALSE.

Description

This function will fetch all the logged in users and return an associative array. The keys of the returned hash will be the user ID numbers; the values will be the users' names. If no users are logged in, FALSE will be returned. The "sort_by", "offset" and "limit" parameters are optional.

A user is considered to be active / logged in if the last access time is within the range of the activity timeout. The timeout can be set using the "ActivityTimeout" directive in a configuration override for "site.ini". The default timeout is one hour.

The "sort_by" parameter must be specified as an array. Each element of the array must be another array where the first element denotes the field (as a string) that the sorting mechanism should use. The second element specifies the direction of the sort (as a boolean). The following sorting fields can be used:

- user_id
- login
- activity

- email

Examples

Example 1

```
{def $users=fetch( 'user', 'logged_in_list',
                  hash( 'sort_by', array( array( 'login', true() ) ) ) )

{foreach $users as $user}
    {$user} <br />
{/foreach}
```

Outputs the names of the users that are currently logged in (sorted by usernames).

logged_in_users

Summary

Fetches the users that are logged in.

Usage

```
fetch( 'user', 'logged_in_users',
      hash( [ 'sort_by', sort_by,
              [ 'offset', offset,
              [ 'limit', limit ] ] ] ) )
```

Parameters

Name	Type	Description	Required
sort_by	mixed	The field that should be used by the sorting mechanism.	No.
offset	integer	The offset to start at.	No.
limit	integer	The number of users that should be fetched.	No.

Returns

An array with ezuser (page 765) objects or FALSE.

Description

This function will fetch all the logged in users and return an array containing ezuser (page 765) objects. If no users are logged in, FALSE will be returned. The "sort_by", "offset" and "limit" parameters are optional.

A user is considered to be active / logged in if the last access time is within the range of the activity timeout. The timeout can be set using the "ActivityTimeout" directive in a configuration override for "site.ini". The default timeout is one hour.

The "sort_by" parameter must be specified as an array. Each element of the array must be another array where the first element denotes the field (as a string) that the sorting mechanism should use. The second element specifies the direction of the sort (as a boolean). The following sorting fields can be used:

- user_id
- login
- activity

- email

Examples

Example 1

```
{def $users=fetch( 'user', 'logged_in_users',
                  hash( 'sort_by', array( array( 'login', true() ) ) ) )

{foreach $users as $user}
    {$user.contentobject.name} <br />
{/foreach}
```

Outputs the names of the users that are currently logged in (sorted by usernames).

member_of

Summary

Fetches the roles that are assigned to a user.

Usage

```
fetch( 'user', 'member_of', hash( 'id', id ) )
```

Parameters

Name	Type	Description	Required
id	integer	The ID number of the target user.	Yes.

Returns

An array with ezrole (page 759) objects or FALSE.

Description

This function will fetch the roles that are assigned to a user. The desired user's ID number must be specified using the "id" parameter. The function will return an array of ezrole (page 759) objects. If no roles are associated with the user, or if an invalid user ID is provided, the function will return FALSE.

Examples

Example 1

```
{def $roles=fetch( 'user', 'member_of', hash( 'id', 42 ) )}

{foreach $roles as $role}
    {$role.name} <br />
{/foreach}
```

Outputs the names of the roles that are assigned to user number 42.

user_role

Summary

Fetches the policies that are available for a user.

Usage

```
fetch( 'user', 'user_role', hash( 'user_id', user_id ) )
```

Parameters

Name	Type	Description	Required
user_id	integer	The user to fetch policies from	Yes.

Returns

An array of hashes or FALSE.

Description

This function will fetch the policies that are available for a user. The desired user's ID number must be specified using the "id" parameter. The function will return an array of policy structures or FALSE if no policies are available or if a non-existing user ID number is provided. The following table shows the structure of the hashes that make up the elements of the returned array.

Name	Type	Description
moduleName	string	The name of the module that the user has access to (* means all modules).
functionName	string	The name of the function that the user has access to (* means all functions).
limitation	string	The elements of the module and function that the user has access to (* means no limitations).

Examples

Example 1

```
{def $policies=fetch( 'user', 'user_role', hash( 'user_id', 42 ) )}

{foreach $policies as $policy}
    {$policy.moduleName} /
    {$policy.functionName} /
    {$policy.limitation} <br />
{/foreach}
```

Outputs information about the policies that are available for user number 42.

Views

activate (page 674)

Provides an interface for activating a user account.

forgotpassword (page 675)

Provides an interface for situations where a user forgets his/her password.

login (page 676)

Provides an interface for logging in a user.

logout (page 677)

Provides a mechanism that logs out a user.

password (page 678)

Provides an interface for changing the password for the current user.

preferences (page 679)

Provides an interface for managing the preferences of the current user.

register (page 680)

Provides an interface for registering a new user.

setting (page 681)

Provides an interface for tweaking user account settings.

success (page 682)

Provides an interface that is called upon a successful user registration.

activate

Summary

Provides an interface for activating a user account.

forgotpassword**Summary**

Provides an interface for situations where a user forgets his/her password.

login**Summary**

Provides an interface for logging in a user.

logout

Summary

Provides a mechanism that logs out a user.

password**Summary**

Provides an interface for changing the password for the current user.

preferences

Summary

Provides an interface for managing the preferences of the current user.

register

Summary

Provides an interface for registering a new user.

setting

Summary

Provides an interface for tweaking user account settings.

success

Summary

Provides an interface that is called upon a successful user registration.

5.3.22 workflow

Summary

Provides views for managing workflows, workflow groups, workflow events, etc.

Description

This module provides an interface to the workflow engine inside the eZ Publish kernel. A workflow is a sequential list of events that is started by a trigger. This module contains views that make it possible to manipulate workflow groups, workflows and events. The administration interface makes use of the views that this module provides in order to allow the users to manage workflows (add new, remove, edit, etc.). Please refer to the "Workflows" (page 163) section of the "Concepts an basics" chapter for more information about workflows.

The module components are documented in the following sections:

- Views (page 684)

Views

down (page 685)

Provides an interface for moving an event to a lower position.

edit (page 686)

Provides an interface for editing a workflow.

event (page 687)

Not documented yet.

groupedit (page 688)

Provides an interface for editing a workflow group.

grouplist (page 689)

Provides an interface for generating a list of all available workflow groups.

process (page 690)

Not documented yet.

run (page 691)

Not documented yet.

up (page 692)

Provides an interface for moving an event to a higher position.

view (page 693)

Provides an interface for viewing a workflow.

workflowlist (page 694)

Provides an interface for generating a list of workflows that belong to a group.

down

Summary

Provides an interface for moving an event to a lower position.

edit

Summary

Provides an interface for editing a workflow.

event

Summary

Not documented yet.

groupedit**Summary**

Provides an interface for editing a workflow group.

grouplist**Summary**

Provides an interface for generating a list of all available workflow groups.

process

Summary

Not documented yet.

run

Summary

Not documented yet.

up

Summary

Provides an interface for moving an event to a higher position.

view

Summary

Provides an interface for viewing a workflow.

workflowlist**Summary**

Provides an interface for generating a list of workflows that belong to a group.

5.4 Views

The views are documented in the following sections:

- class (page [373](#))
- collaboration (page [392](#))
- content (page [407](#))
- error (page [527](#))
- ezinfo (page [528](#))
- form (page [533](#))
- infocollector (page [536](#))
- layout (page [541](#))
- notification (page [546](#))
- package (page [558](#))
- pdf (page [581](#))
- reference (page [585](#))
- role (page [586](#))
- rss (page [594](#))
- search (page [600](#))
- section (page [603](#))
- setup (page [617](#))
- shop (page [618](#))
- trigger (page [646](#))
- url (page [649](#))
- user (page [658](#))
- workflow (page [683](#))

5.5 Objects

ezauthor (page 699)

Contains information about authors.

ezbasket (page 700)

Contains information about a user's shopping basket.

ezbinaryfile (page 702)

Contains information about a file.

ezcontentbrowsebookmark (page 703)

Contains information about a bookmark.

ezcontentbrowserecent (page 704)

Contains information about a node with recently edited children.

ezcontentclass (page 705)

Contains information about a content class.

ezcontentclassattribute (page 708)

Contains information about an attribute of a content class.

ezcontentclassclassgroup (page 710)

Contains information about a class group assignment.

ezcontentclassgroup (page 711)

Contains information about a class group.

ezcontentobject (page 712)

Contains information about a content object.

ezcontentobjectattribute (page 717)

Contains information about an attribute of a content object.

ezcontentobjecttranslation (page 720)

Contains information about a translation.

ezcontentobjecttreenode (page 721)

Contains information about a node within the content node tree.

ezcontentobjectversion (page 725)

Contains information about a version of a content object.

ezdate (page 728)

Contains information about a date.

ezdatetime (page 729)

Contains information about a date and time.

ezimagealiashandler (page 730)

Contains information about an image.

ezimagedlayer (page 734)

Contains information about an image layer.

ezimageobject (page 735)

Contains information about an image.

ezinformationcollection (page 736)

Contains information about a block of collected information.

ezinformationcollectionattribute (page 737)

Contains information about an attribute of a collection.

ezkeyword (page 738)

Contains information about keywords.

ezlocale (page 739)

Contains information about a locale.

ezmatrix (page 742)

Contains information about a matrix.

ezmedia (page 745)

Contains information about a video file.

ezmultioption (page 747)

Contains information about multilpe options.

eznodeassignment (page 748)

Contains information about a node assignment.

ezoption (page 750)

Contains information about a collection of options.

ezorder (page 751)

Contains information about an order.

ezorderstatus (page 754)

Contains information about an order status.

ezpolicy (page 755)

Contains information about a policy.

ezprice (page 756)

Contains information about a price.

ezproductcollectionitem (page 757)

Contains information about an item of a product collection.

ezrangeoption (page 758)

Contains information about a range of options.

ezrole (page 759)

Contains information about a role.

ezsection (page 760)

Contains information about a section.

ezsimplifiedxmlinput (page 761)

Contains information about XML data.

ezsubtreenotificationrule (page 762)

Contains information about a subtree notification rule.

eztime (page 763)

Contains information about a time value.

ezurl (page 764)

Contains information about a URL.

ezuser (page 765)

Contains information about a user.

ezvattype (page 767)

Contains information about a VAT.

ezxhtmlxmloutput (page 768)

Contains information about XML data.

ezxmlinpushandler (page 769)

Contains information about XML data.

ezxmloutpushandler (page 770)

Contains information about XML data.

ezxmltext (page 771)

Contains information about an XML block.

5.5.1 ezauthor

Summary

Contains information about authors.

Attributes

Attribute	Type	Description
author_list	array	Contains information about the authors. Each element in the array consists of a hash of strings. The keys are "id", "name" and "email".
name	NULL	Not used.
is_empty	boolean	Returns TRUE if the object does not contain any authors (if the "author_list" array is empty); otherwise FALSE is returned.

5.5.2 ezbasket

Summary

Contains information about a user's shopping basket.

Attributes

Attribute	Type	Description
id	string	The ID number of the shopping basket.
session_id	string	The ID of the session that the basket belongs to.
productcollection_id	string	The ID number of the product collection that belongs to the basket.
order_id	string	The ID number of the order that belongs to the basket.
items	array	An array of hashes containing information about the items. Each element consists of the following data: Attribute: id Type: string Description: The ID number of the item in the basket. Attribute: vat_value Type: string Description: The actual value of VAT (for example, 22). Attribute: item_count Type: string Description: The quantity. Attribute: node_id Type: string Description: The ID number of the node that represents the item. Attribute: object_name Type: string

		<p>Description: The name of the object that represents the item.</p>
		<p>Attribute: price_ex_vat Type: double Description: The price of the item excluding the VAT.</p>
		<p>Attribute: price_inc_vat Type: string Description: The price of the item including the VAT.</p>
		<p>Attribute: discount_percent Type: string Description: The discount percentage (if any).</p>
		<p>Attribute: total_price_ex_vat Type: double Description: The total price excluding the VAT.</p>
		<p>Attribute: total_price_inc_vat Type: double Description: The total price including the VAT.</p>
		<p>Attribute: item_object Type: object Description: The item itself (as an ezproductcollectionitem (page 757) object).</p>
total_ex_vat	float	The total amount to be payed excluding the VAT.
total_inc_vat	float	The total amount to be payed including the VAT.
is_empty	boolean	Returns TRUE if there are no items in the basket, FALSE otherwise.

5.5.3 ezbinaryfile

Summary

Contains information about a file.

Attributes

Attribute	Type	Description
contentobject_attribute_id	string	The ID number of the content object attribute that the file belongs to.
version	string	The version number of the object that the file belongs to.
filename	string	The internal name of the file (generated by the system).
original_filename	string	The original name of the file.
mime_type	string	The MIME type of the file (for example "audio/wav").
download_count	string	The number of times the file has been downloaded through the "download" (page 504) view of the "content" module.
filesize	integer	The size of the file (number of bytes).
filepath	string	The path to the file (including the filename).
mime_type_category	string	The MIME type category (for example "audio").
mime_type_part	string	The MIME type part (for example "wav").

5.5.4 ezcontentbrowsebookmark

Summary

Contains information about a bookmark.

Attributes

Attribute	Type	Description
id	string	The ID number of the bookmark.
user_id	string	The ID number of the user that the bookmark belongs to.
node_id	string	The ID number of the bookmarked node.
name	string	The name of the bookmark (the same as the name of the node).
node	object	The bookmarked node (as ezcontentobjecttreenode (page 721) object).
contentobject_id	string	The ID number of the object that is referenced by the bookmarked node.

5.5.5 ezcontentbrowserecent

Summary

Contains information about a node with recently edited children.

Attributes

Attribute	Type	Description
id	string	A unique ID number.
user_id	string	The ID number of the user that the "browse recent" entry belongs to.
node_id	string	The ID number of the node under which something was recently published.
created	string	A UNIX timestamp pinpointing the exact date/time when the "browse recent" entry was created.
name	string	The name of the node under which something was recently published.
node	object	The actual node under which something was recently published (as an ezcontentobjecttreenode (page 721) object).
contentobject_id	string	The ID of the object which is encapsulated by the node under which something was recently published.

5.5.6 ezcontentclass

Summary

Contains information about a content class.

Attributes

Attribute	Type	Description
id	string	The ID number of the class.
version	string	The version/status of the class (0=normal, 1=temporary, 2=modified).
name	string	The name of the class (for example "News article").
identifier	string	The identifier of the class (for example "news_article").
contentobject_name	string	The pattern which controls how the names of the instances should be generated.
creator_id	string	The ID number of the object that represents the user who created the class.
modifier_id	string	The ID number of the object that represents the user who most modified the class last.
created	string	A UNIX timestamp pinpointing the exact date/time when the class was created.
remote_id	string	A global unique ID for the class. The remote ID is an MD5 hash of the time when the class was created plus a random value. Remote IDs are used to avoid collision of identical classes during an import.
modified	string	A UNIX timestamp pinpointing the exact date/time when the class was last modified.
is_container	string	Either 1 or 0. Reveals whether nodes referencing objects of this class should be considered as containers or not. Used by the administra-

		tion interface to allow or disallow the creation of nodes under a node which references an object of this class.
data_map	array	The attributes (as ezcontent-classattribute (page 708) objects) that make up the class.
object_count	string	The number of instances (objects) of the class.
version_count	string	DEPRECATED - Similar to the "version" attribute, but will be 2 if the "version_count" is 2 or higher.
version_status	string	DEPRECATED - The version count of the class if it has been determined, FALSE if not determined.
ingroup_list	array	The class groups (as ezclass-classgroup (page 710) objects) that the class is a member of.
ingroup_id_list	array	The class groups (as ezclass-group (page 710) objects) that the class belongs to.
match_ingroup_id_list	array	The class groups (as ezclassgroup (page 710) objects) that the class belongs to. This variable is connected with a feature that makes it possible to create template overrides based on class groups. By default the "match_ingroup_id_list" always returns a boolean FALSE value because the class group override feature is turned off. It can be turned on by setting the "EnableClassGroupOverride" directive in the [ContentOverrideSettings] block of a configuration override for "content.ini" to "true".
group_list	array	All the class groups (as ez-

		classclassgroup (page 710) objects) that are present in the database.
creator	object	The object (as ezcontentobject (page 712) object) representing the user who created the class.
modifier	object	The object (as ezcontentobject (page 712) object) representing the user who last modified the class.

5.5.7 ezcontentclassattribute

Summary

Contains information about an attribute of a content class.

Attributes

Attribute	Type	Description
id	string	The ID number of the class attribute.
name	string	The name of the class attribute.
version	string	The version number of the version that the attribute belongs to.
contentclass_id	string	The ID number of the class that the attribute belongs to.
identifier	string	The identifier of the class attribute (for example "first_name").
placement	string	The location of the class attribute within the list of attributes.
is_searchable	string	Either 1 or 0. 1 means that the content stored using this attribute will be indexed by the search engine, 0 means that the content will not be indexed.
is_required	string	Either 1 or 0. 1 means that input is required, 0 means that empty inputs are allowed.
can_translate	string	Either 1 or 0. 1 means that instances of the attribute can be translated to different languages. 0 means that no translations (except for the default translation) can be made.
is_information_collector	string	Either 1 or 0. 1 means that the attribute functions as an information collector. 0 means that the attribute is

		just a normal attribute and thus stores data in the default/normal way.
data_type_string	string	The identifier string of the datatype that is used to represent the class attribute (for example "ezstring").
data_int1	string	Integer 1.
data_int2	string	Integer 2.
data_int3	string	Integer 3.
data_int4	string	Integer 4.
data_float1	string	Float 1.
data_float2	string	Float 2.
data_float3	string	Float 3.
data_float4	string	Float 4.
data_text1	string	Text 1.
data_text2	string	Text 2.
data_text3	string	Text 3.
data_text4	string	Text 4.
content	mixed	Data for the datatype which this class attribute is made of, the actual data depends on the datatype.
temporary_object_attribute	object	A temporary content object attribute (as ezcontentobjetc-tattribute (page 717) object) which does not exist in the database..
datatype	object	The datatype that is used to represent the class attribute.
display info	array	Array of miscellaneous display parameters used by the system (for example whether the components of the edit interface should be grouped or not).

5.5.8 ezcontentclassclassgroup

Summary

Contains information about a class group assignment.

Attributes

Attribute	Type	Description
contentclass_id	string	The ID number of the class which belongs to the group.
contentclass_version	string	The version (either 1 or 0).
group_id	string	The ID number of the class.
group_name	string	The name of the class group (for example "Media").

5.5.9 ezcontentclassgroup

Summary

Contains information about a class group.

Attributes

Attribute	Type	Description
id	string	The ID number of the class group.
name	string	The name of the class group (for example "Media").
creator_id	string	The ID number of the object representing the user who created the class group.
modifier_id	string	The ID number of the object representing the user who last modified the class group.
created	string	A UNIX timestamp pinpointing the exact date/time when the class group was created.
modified	string	A UNIX timestamp pinpointing the exact date/time when the class group was last modified.

5.5.10 ezcontentobject

Summary

Contains information about a content object.

Attributes

Attribute	Type	Description
id	string	The ID number of the object.
section_id	string	The ID number of the section that the object belongs to.
owner_id	string	The ID number of the object representing the user who initially created the object.
contentclass_id	string	The ID number of the content class which the object is an instance of.
name	string	The actual name of the object (for example "Liver sandwich").
is_published	string	Either 1 or 0. 1 means that the object has been published. 0 means that the object has not yet been published.
published	string	A UNIX timestamp pinpointing the exact date/time when the object was published for the first time.
modified	string	A UNIX timestamp pinpointing the exact date/time when the object was last modified.
current_version	string	The number of the currently published version.
status	string	The status of the object (0=Draft, 1=Published, 2=Archived).
remote_id	string	A global unique ID for the object. The remote ID is an MD5 hash of the time when the object was created plus a random value. Remote IDs are used in order to avoid collisions of identical objects

		during an import.
current	object	The current version (as ezcontentobjectversion (page 725)) of the object.
versions	array	The object's versions (as ezcontentobjectversion (page 725) objects).
author_array	array	Array of ezuser (page 765) objects representing the different creators of the content object's versions.
class_name	string	The name of the class which the content object is an instance of (for example "Consumer product").
content_class	object	The content class (as ezcontentclass (page 705)) which the content object is an instance of.
contentobject_attributes	array	Array of ezcontentobjectattribute (page 717) objects representing the attributes of the content object.
owner	object	An ezcontentobject (page 712) that represents the user who initially created the object.
related_contentobject_array	array	An array of ezcontentobject (page 712) objects that are related to this object. This attribute is deprecated. It is recommended to use the "related_objects (page 465)" fetch function instead.
related_contentobject_count	string	The number of objects that are related to this object. This attribute is deprecated. It is recommended to use the "related_objects_count (page 468)" fetch function instead.
reverse_related_contentobject_array	array	An array of ezcontentobject (page 712) objects that make use of this object (reverse relations). This attribute is deprecated. It is recommended to use the "reverse_

		related_objects (page 469)” fetch function instead.
reverse_related_contentobject_count	string	The number of objects that are using this object. This attribute is deprecated. It is recommended to use the “reverse_related_objects_count (page 472)” fetch function instead.
can_read	boolean	Returns TRUE if the current user has read access to the object, FALSE otherwise.
can_edit	boolean	Returns TRUE if the current user has edit access to the object, FALSE otherwise.
can_translate	boolean	Returns TRUE if the current user has permissions to translate the contents of the object, FALSE otherwise.
can_move_from	boolean	Returns TRUE if the current user has permissions to move the main node of the object, FALSE otherwise.
can_remove	boolean	Returns TRUE if the current user has permissions to remove the object, FALSE otherwise.
data_map	array	Array of ezcontentobjectattribute (page 717) objects representing the actual attributes of the content object.
main_parent_node_id	string	The ID number of the main node of the object encapsulated by the parent node.
assigned_nodes	array	Array of nodes (as ezcontentobjecttreenode (page 721) objects) that encapsulate the object.
parent_nodes	array	An array of ID numbers of the parent nodes (as strings).
main_node_id	string	The ID number of the object's main node.
main_node	object	The object's main node (as ezcontentobjecttreenode (page 721)).
default_language	string	The default language of the

		object (for example "eng-GB").
content_action_list	array	An array of hashes revealing information about the content actions that can be performed on the object. The keys "name" and "action" contain the actual name (for example "Add to basket" - which should be value of the HTML input tag) and the action itself (for example "ActionAddToBasket" - which should be the name of the HTML input tag). The array is generated by a function that examines the object's attributes. If a datatype used to represent an attribute provides support for content actions or if the attribute is an information collector, the supported actions will be added to the "content_action_list" array. This array can be used to automatically generate action buttons (standard HTML buttons) for content objects that either make use of special datatypes or have attributes that are marked as information collectors.
class_identifier	string	The identifier of the class which the object is an instance of (for example "consumer_product").
class_group_id_list	array	An array of ID numbers of the class groups which the class (that the object is an instance of) belongs to.
matchingroup_id_list	array	The ID numbers of the class groups that the class (which the object is an instance of) belongs to. This variable is connected with a feature that makes it possible to create

template overrides based on class groups.

By default the "matchingroup_id_list" always returns a boolean FALSE value because the class group override feature is turned off. It can be turned on by setting the "EnableClassGroupOverride" directive in the [ContentOverrideSettings] block of a configuration override for "content.ini" to "true".

5.5.11 ezcontentobjectattribute

Summary

Contains information about an attribute of a content object.

Attributes

Attribute	Type	Description
id	string	The ID number of the attribute.
contentobject_id	string	The ID number of the content object that the attribute belongs to.
version	string	The version number of the content object that the attribute belongs to.
language_code	string	The code of the translation that the attribute belongs to (for example "eng-GB").
contentclassattribute_id	string	The ID number of the attribute.
attribute_original_id	string	The original ID of the attribute.
sort_key_int	string	Integer used for sorting.
sort_key_string	string	Text used for sorting.
data_type_string	string	The identifier string of the datatype (for example "ezstring").
data_text	string	Text stored by the attribute.
data_int	string	Integer stored by the attribute.
data_float	string	Float stored by the attribute.
contentclass_attribute	object	The class attribute (as an ezcontentclassattribute (page 708) object).
contentclass_attribute_identifier	string	The identifier of the content class attribute (for example "first_name").
contentclass_attribute_name	string	The name of the content class attribute.
can_translate	string	1 if the attribute is translatable, 0 if not.
is_information_collector	string	1 if the attribute is an infor-

		mation collector, 0 if not.
is_required	string	1 if the attribute is required, 0 if not.
content	any	The actual content (what is returned when the ".content" notation is used).
has_http_value	boolean	TRUE if the attribute has an HTTP value, FALSE otherwise.
value	any	The HTTP input from the user (if submitted) or the contents of the object attribute from the database (same as the "content" attribute).
has_content	boolean	TRUE if there is attribute contains content, FALSE if it is empty.
class_content	any	The content of the class attribute which this object attribute is made from (same as ".contentclass_attribute.content").
object	object	The object that the attribute belongs to (as an ezcontentobject (page 712) object).
view_template	string	The name of the template that is used to display the view interface for the attribute (for example "ezstring").
edit_template	string	The name of the template that is used to display the edit interface for the attribute (for example "ezstring").
result_template	string	The name of the template that is used to display the information that was collected by the attribute (for example "ezstring").
has_validation_error	boolean	TRUE if a validation error was detected, FALSE if everything is okay.
validation_error	NULL	The validation error(s),

		NULL if none.
validation_log	NULL	A log of the validation error(s), NULL if none.
language	object	The original translation (as ezcontentobjectattribute (page 717) object) of this content object attribute (it may just be the same object).
is_a	string	Returns the identifier of the datatype that is used to represent the attribute (for example "ezstring").
display_info	array	An array containing information about how the attribute should be displayed in different scenarios (for example if the information should be grouped, etc.).
class_display_info	array	An array containing information about how the attribute should be displayed on the class level.

5.5.12 ezcontentobjecttranslation

Summary

Contains information about a translation.

Attributes

Attribute	Type	Description
contentobject_id	string	The ID number of the object that the translation belongs to.
version	string	The version number that the translation belongs to.
language_code	string	The translation's language code (for example "eng-GB").
locale	object	The locale (as ezlocale (page 739) object) that the translation uses.

5.5.13 ezcontentobjecttreenode

Summary

Contains information about a node within the content node tree.

Attributes

Attribute	Type	Description
node_id	string	The ID number of the node.
parent_node_id	string	The ID number of the parent node.
main_node_id	string	The ID number of the main node.
contentobject_id	string	The ID number of the content object.
contentobject_version	string	The number of the published version.
contentobject_is_published	string	The published status of the object (0=not published, 1=published).
depth	string	The depth of the node within the content node tree. The depth of a top level node is 1.
sort_field	string	The sorting method used to sort the child nodes.
sort_order	string	The sorting order used when sorting the node's children.
priority	string	The node's priority (positive or negative integer).
modified_subnode	string	A UNIX timestamp pinpointing the exact time a sub node was changed.
path_string	string	The node's path string (for example "/1/2/44").
path_identification_string	string	The node's path identification string (for example "company/about_us").
remote_id	string	A unique ID for the node (avoids crashes when importing/exporting nodes). A remote ID is an MD5 hash of the time when the node was generated plus a random

		value.
is_hidden	string	The node's hidden status (0=visible, 1=hidden).
is_invisible	string	The node's visibility status (0=visible, 1=hidden by superior).
name	string	The name of the object the node encapsulates (for example "My article").
data_map	array	The object's attributes as ezcontentobjectattribute (page 717) objects.
object	object	The actual content object (as ezcontentobject (page 712)) that the node encapsulates.
subtree	array	All the nodes that are below this node as ezcontentobjecttreenode (page 721) objects.
children	array	Array of nodes that are directly below this node as ezcontentobjecttreenode (page 721) objects.
children_count	string	The number of nodes that are directly below this node.
contentobject version object	object	The current version (as ezcontentobjectversion (page 725)) of the object that the node encapsulates.
sort_array	array	The node's sort array.
can_read	boolean	Returns TRUE if the current user has read access to the node (FALSE otherwise).
can_create	boolean	Returns TRUE if the current user can create nodes under this node (FALSE otherwise).
can_edit	boolean	Returns TRUE if the current user has edit access to the node (FALSE otherwise).
can_hide	boolean	Returns TRUE if the current user can modify the hidden state of the node (FALSE otherwise).
can_remove	boolean	Returns TRUE if the current user can remove the node (FALSE otherwise).
can_move	boolean	Returns TRUE if the current

		user can move the node to another location (FALSE otherwise).
can_move_from	boolean	Same as "can_move", returns TRUE if the current user has permissions to move node, FALSE otherwise.
creator	object	The object (as ezcontentobject (page 712)) containing the user who created the node.
path	array	Array containing the nodes that make up the path as ezcontentobjecttreenode (page 721) objects. The current node is not included.
path_array	array	Array of strings revealing the ID numbers of the nodes that make up the path. The current node is also included.
parent	object	The parent node (as ezcontentobjecttreenode (page 721)).
url	string	The URL of the node either as a virtual URL ("company/about_us") or a system URL ("content/view/full/13") depending on a configuration setting.
url_alias	string	The virtual URL of the node ("company/about_us").
class_identifier	string	The identifier of the class which the object encapsulated by the node is an instance of (for example "product_review").
class_name	string	The name of the class which the object encapsulated by the node is an instance of (for example "Product review").
hidden_invisible_string	string	The visibility status of a node ("/-" = completely visible, "H/X" = hidden by user and thus invisible, "-/X" = hid-

hidden_status_string	string	den by superior). The visibility status of the node: "Visible", "Hidden" or "Hidden by superior".
----------------------	--------	--

5.5.14 ezcontentobjectversion

Summary

Contains information about a version of a content object.

Attributes

Attribute	Type	Description
id	string	The ID number of the version.
contentobject_id	string	The ID number of the object that the version belongs to.
creator_id	string	The ID number of the object that represents the user who created the version.
version	string	The actual version number.
status	string	The status of the version.
created	string	A UNIX timestamp pinpointing the exact date/time when the version was created.
modified	string	A UNIX timestamp pinpointing the exact date/time when the version was last modified.
workflow_event_pos	string	DEPRECATED - was related to workflows.
user_id	string	DEPRECATED - was related to workflows.
creator	object	The object (as ezcontentobject (page 712) object) that represents the user who created the version.
name	string	The name of the version (generated using the object name pattern).
version_name	string	The name of the version (generated using the object name pattern).
main_parent_node_id	string	The ID number of the main parent node that references the object which the version belongs to.
contentobject_attributes	array	The attributes (as ezcontentobjectattribute (page

		717) objects) that the version consists of.
related_contentobject_array	array	An array of ezcontentobject (page 717) objects representing the objects that are related to the object that the version belongs to.
reverse_related_object_list	array	An array of ezcontentobject (page 717) objects representing the objects that are related to the object which the version belongs to.
parent_nodes	array	The parent nodes (as eznodeassignment (page 748) objects) of the nodes which reference the object that the version belongs to.
can_read	boolean	Returns TRUE if the current user has read access to the version. Otherwise FALSE is returned.
data_map	array	A hash containing the attributes (as ezcontentobjectattribute (page 717) objects) that the version consists of. The keys of the hash are the identifiers of the attributes.
node_assignments	array	An array of node assignments (as eznodeassignment (page 748) objects) that are connected with the object which the version belongs to.
contentobject	object	The object (as ezcontentobject (page 712)) that the version belongs to.
language_list	array	The translations (as ezcontentobjecttranslation (page 720) objects) that belong to the version - including the default translation.
translation	object	The default translation (as ezcontentobjecttranslation (page 720) object).
translation_list	array	The translations (as ezcontentobjecttransla-

		tion (page 720) objects) that belong to the version - the default translation is not included.
complete_translation_list	array	Same as the "translation_list" attribute.
temp_main_node	object	A temporary node (as ezcontentobjecttreenode (page 721) object) for the object that the version belongs to. The temporary node does not exist in the database.

5.5.15 ezdate

Summary

Contains information about a date.

Attributes

Attribute	Type	Description
timestamp	string	The date as a UNIX timestamp (for example "567990000") if the date is a valid date, NULL otherwise.
is valid	boolean	Returns TRUE if the date is a valid date, FALSE otherwise.
year	string	The year (for example "1988").
month	string	The month (for example "01").
day	string	The day (for example "01").

5.5.16 ezdatetime

Summary

Contains information about a date and time.

Attributes

Attribute	Type	Description
timestamp	string	The date/time value as a UNIX timestamp (for example "1147719660") if the date/time is valid.
hour	string	The hour (for example "21").
minute	string	The minute (for example "01").
year	string	The year (for example "2006").
month	string	The month (for example "05").
day	string	The day (for example "15").
is_valid	boolean	Returns TRUE if the date/time value is valid, FALSE otherwise.

5.5.17 ezimagealiashandler

Summary

Contains information about an image.

Attributes

Attribute	Type	Description
alternative_text	string	The alternative image text (for example "Picture of an apple.").
original_filename	string	The original name of the image file (for example "apple.png").
is_valid	string	Returns either 1 or 0 (valid or invalid).
name of variation	array	Attribute: name Type: string Description: The name of the variation (for example "original"). Attribute: width Type: string Description: The width as number of pixels (for example "320"). Attribute: height Type: string Description: The height as number of pixels (for example "256"). Attribute: mime_type Type: string Description: The MIME type (for example "image/png"). Attribute: filename Type: string Description: The name of the file (for example "my_image.png").

	<p>Attribute: suffix Type: string Description: The file suffix (for example "png").</p>
	<p>Attribute: dirpath Type: string Description: The path to the image (for example "var/storage/images/test/199-2-eng-GB").</p>
	<p>Attribute: basename Type: string Description: The basename of the image file (for example "apple").</p>
	<p>Attribute: alternative_text Type: string Description: The alternative image text (for example "Picture of an apple."). </p>
	<p>Attribute: text Type: string Description: Contains the "alternative_text" of the original image.</p>
	<p>Attribute: original_filename Type: string Description: The name of the original file (for example "apple.png").</p>
	<p>Attribute: url Type: string Description: Complete path + name of image file (for example "var/storage/images/test/199-2-eng-GB/apple.png").</p>
	<p>Attribute: alias_key</p>

Store your information

Type: string
Description: A internal CRC32 value which is used when an alias is created. This value is based on the filters that were used (parameters included) and is checked when an alias is accessed. If this values differs from the configured filters (in image.ini or an override), the system will recreate the alias.

Attribute: timestamp
Type: string
Description: A UNIX timestamp pinpointing the exact date/time when the alias was last modified. For the "original" alias, the timestamp will reveal the time when the image was originally uploaded.

Attribute: full_path
Type: string
Description: Complete path + name of image file (for example "var/storage/images/test/199-2-eng-GB/apple.png").

Attribute: is_valid
Type: string
Description: TRUE if the alias was created properly, that means all conversion and filters were applied without problems. It will be FALSE if the image manager is wrongly configured or if there no compatible image converters could be found.

Attribute: is_new
Type: boolean

		<p>Description: Will be set to TRUE the first time the alias is created, the next time (reload of the same page) it will be FALSE. It will also be set to TRUE every time the alias is re-created due to changes in filters (see alias_key).</p> <p>Attribute: filesize</p> <p>Type: integer</p> <p>Description: The number of bytes that the image consumes.</p> <p>Attribute: info</p> <p>Type: string</p> <p>Description: Contains extra information about the image, depending on the image type. It will typically contain EXIF information from digital cameras or information about animated GIFs. If there is no information, the info will be a boolean FALSE.</p>
--	--	--

5.5.18 ezimagerlayer

Summary

Contains information about an image layer.

Attributes

Attribute	Type	Description
filepath	string	The path (for example "design/example/images").
filename	string	The name of the image file (for example "delorean.png").
width	integer	The width of the image (number of pixels).
height	integer	The height of the image (number of pixels).
alternative_text	string	The alternative image text.
imagepath	string	The internal (eZ publish) path to the image (for example "design/example/images/delorean.png").
has_size	boolean	TRUE if "width" and "height" is set, otherwise FALSE.

5.5.19 ezimageobject

Summary

Contains information about an image.

Attributes

Attribute	Type	Description
filepath	string	The path (for example "design/example/images").
filename	string	The name of the image file (for example "delorean.png").
width	integer	The width of the image (number of pixels).
height	integer	The height of the image (number of pixels).
alternative_text	string	The alternative image text.
imagepath	string	The internal (eZ publish) path to the image (for example "design/example/images/delorean.png").
has_size	boolean	TRUE if "width" and "height" is set, otherwise FALSE.

5.5.20 ezinformationcollection

Summary

Contains information about a block of collected information.

Attributes

Attribute	Type	Description
id	string	The ID number of the information collection.
contentobject_id	string	The ID number of the object that collected the information.
user_identifier	string	The identifier of the user that submitted the information.
created	string	A UNIX timestamp revealing the exact date/time when the information was collected.
modified	string	A UNIX timestamp revealing the exact date/time when the collection was last modified.
attributes	array	An array of the collection attributes (as ezinformationcollectionattribute (page 737) objects).
object	object	The actual object that collected the information (as an ezcontentobject (page 712) object).

5.5.21 ezinformationcollectionattribute

Summary

Contains information about an attribute of a collection.

Attributes

Attribute	Type	Description
id	string	The ID number of the information collection attribute.
informationcollection_id	string	The ID number of the information collection itself.
contentclass_attribute_id	string	The ID number of the class attribute.
contentobject_attribute_id	string	The ID number of the object attribute.
contentobject_id	string	The ID number of the object.
data_text	string	Collected text.
data_int	string	Collected integer.
data_float	string	Collected float.
contentclass_attribute_name	string	The name of the attribute that collected the information.
contentclass_attribute	object	The class attribute (as an ezcontentclassattribute (page 708) object).
contentobject_attribute	object	The object attribute (as an ezcontentobjectattribute (page 717) object).
contentobject	object	The content object (as an ezcontentobject (page 712) object).
result_template	string	The name of the result template (for example "ezstring").

5.5.22 ezkeyword

Summary

Contains information about keywords.

Attributes

Attribute	Type	Description
keywords	array	An array of strings containing the keywords/phrases.
keyword_string	string	The actual keyword string (comma separated keywords/phrases).
related_nodes	array	An array of nodes (as ezcontentobjecttreenode (page 721) objects) that have at least one of the same keywords.
related_objects	array	(Deprecated from 3.6.1, use related_nodes) An array of nodes (as ezcontentobjecttreenode (page 721) objects) that have at least one of the same keywords.

5.5.23 ezlocale

Summary

Contains information about a locale.

Attributes

Attribute	Type	Description
charset	string	The character set that the locale uses (for example "iso-8859-1").
allowedCharsets	array	An array of strings containing the allowed character sets.
country_name	string	The name of the country that the locale belongs to (for example "United Kingdom").
country_comment	string	A comment about the country that the locale belongs to (usually empty).
country_code	string	The country code (for example "GB" for Great Britain).
country_variation	string	The country variation.
language_name	string	The native name of the language (for example "Norsk", "Magyar", etc.).
intl_language_name	string	The international name of the language (for example "Norwegian", "Hungarian", etc.).
language_code	string	The language code (for example "eng").
language_comment	string	A comment about the language itself (usually empty).
locale_code	string	The actual locale code (for example "eng-GB", "nor-NO", etc.).
locale_full_code	string	The full locale code (for example "eng-GB").
http_locale_code	string	The HTTP locale code (for example "eng-GB").
decimal_symbol	string	The decimal symbol (for example a dot ".").
thousands_separator	string	The character (if any) that is

		used to separate/split large numbers.
decimal_count	string	The number of decimal digits that should be displayed.
negative_symbol	string	The symbol used for displaying negative numbers (usually just a dash: "-").
positive_symbol	string	The symbol used for displaying positive numbers (usually empty).
currency_decimal_symbol	string	The symbol used for separating the integer part from the decimal part of currency values.
currency_thousands_separator	string	The thousand separator used for currencies.
currency_decimal_count	string	The number of decimal digits that should be included when displaying currencies.
currency_negative_symbol	string	The symbol used for displaying negative currencies (usually just a dash: "-").
currency_positive_symbol	string	The symbol used for displaying positive currencies (usually empty).
currency_symbol	string	The currency symbol (for example "£").
currency_name	string	The name of the currency (for example "British Pound", "Norwegian Kroner", etc.).
currency_short_name	string	A short/abbreviated name for the currency (for example "BSP", "NOK", etc.).
is_monday_first	boolean	Returns TRUE if monday is considered to be the first day of the week, FALSE otherwise.
weekday_name_list	array	An array of strings containing the weekday names (for example "Monday", "Tuesday", etc.).
weekday_short_name_list	array	An array of strings containing abbreviated weekday names (for example "Mon", "Tue", etc.).
weekday_number_list	array	An array of strings contain-

		ing the weekday numbers (for example "0", "1", etc.).
month_list	array	An array of strings containing the month digits (for example "1" for January, "2" for February, etc.).
month_name_list	array	An array of strings containing the name of the months (for example "January", "February", etc.).
is_valid	boolean	Returns TRUE if the locale is valid (successfully read from disk), FALSE otherwise (unknown locale).

5.5.24 ezmatrix

Summary

Contains information about a matrix.

Attributes

Attribute	Type	Description
name	boolean	Always FALSE (this attribute is currently not used).
rows	array	A collection of miscellaneous structures that contain information about the rows. Currently there is only one structure, called "sequential". It is built up of an array of hashes. The following table reveals the structure of the array elements. Name: identifier Type: string Description: The identifier of the column (defined at the class level). Name: name Type: string Description: The name of the column (defined at the class level). Name: columns Type: array Description: An array of strings holding the actual contents of the columns.
columns	array	A collection of miscellaneous structures that contain information about the rows. Currently there are two types of structures: "sequential" and "id". The "sequential" structure is built up of an array

Some
of your
information

hashes. The following table reveals the structure of the array elements.

Name: identifier

Type: string

Description: The identifier of the column (defined at the class level).

Name: index

Type: string

Description: The row index ("0", "1", and so on).

Name: name

Type: string

Description: The name of the column (defined at the class level).

Name: rows

Type: array

Description: An array of strings holding the actual contents of the rows.

The "id" structure consists of hash where the keys are the column identifiers. The following table shows the structure that is available for each column identifier.

Name: identifier

Type: string

Description: The identifier of the column (defined at the class level).

Name: index

Type: string

Description: The column index ("0", "1", and so on).

Name: name

Type: string

Description: The name of

		the column (defined at the class level).
		Name: rows Type: array Description: An array of strings holding the actual contents of the rows.
cells	array	A flat array of the cells that make up the matrix (from left to right, top to bottom).
matrix	array	Consists of "rows", "columns" and "cells" (see above).
rowCount	integer	The number of rows.
columnCount	integer	The number of columns.

5.5.25 ezmedia

Summary

Contains information about a video file.

Attributes

Attribute	Type	Description
contentobject_attribute_id	string	The identification number of the content object attribute.
version	string	The content object version.
filename	string	The name of the file in the eZ publish var directory (for example "44b963c9e8d1ffa80cbb08e84d576735.avi").
original_filename	string	The original filename (for example "ezpublish-mediademo.avi").
mime_type	string	The MIME type (for example "video/x-msvideo").
width	string	The playback width - in number of pixels (for example "640").
height	string	The playback height - in number of pixels (for example "480").
has_controller	string	Either 1 or 0 (show controller or do not show controller).
controls	string	Either 1 or 0 - Real Media specific - controls the control-bar.
is_autoplay	string	Either 1 or 0 (automatically start playback or not).
pluginspage	string	A URL that leads to the plugin that is required for proper playback.
quality	string	Flash specific - controls the quality of the media.
is_loop	string	Either 1 or 0 (looped playback or single-cycle).
filesize	integer	The number of bytes the media file consumes.
filepath	string	The path to the media file (for example "var/

		storage/original/video/ 44b963c9e8d1ffa80cbb08e84d576735.av
mime_type_category	string	The MIME type category (for example "video").
mime_type_part	string	The MIME type part (for example "x-msvideo").

5.5.26 ezmultioption

Summary

Contains information about multiple options.

Attributes

Attribute	Type	Description
name	string	The name of the entire multi-option set.
multioption_list	array	An array of hashes where each hash consists of the following elements: Attribute: id Type: integer Description: The ID number of the option. Attribute: name Type: string Description: The name of the option (for example "Color"). Attribute: priority Type: string Description: The option's priority. Attribute: default_option_id Type: string Description: The ID number of the default option. Attribute: optionlist Type: array Description: Array of hashes - the structure is described in the documentation page for the ezoption (page 750) object.

5.5.27 eznodeassignment

Summary

Contains information about a node assignment.

Attributes

Attribute	Type	Description
id	string	The ID number of the node assignment.
remote_id	string	The remote ID of the node assignment.
contentobject_id	string	The ID number of the object that the node assignment belongs to.
contentobject_version	string	The version number which the node assignment belongs to.
parent_node	string	The ID number of the parent node.
sort_field	string	The ID number of the method for sorting child nodes.
sort_order	string	Either 1 (ascending) or 0 (descending). Reveals the order for sorting child nodes.
is_main	string	Either 1 or 0. 1 means that this is the main node assignment for the object.
from_node_id	string	The ID number of the original node. This attribute will only have a valid value if an existing node is moved, in which case the "parent_node_id" will reveal the new parent node. If the attribute contains "0" or "-1", it means that the node is not meant to be moved.
parent_remote_id	string	The remote ID of the parent node.
parent_node_obj	object	The parent node (as ezcontentobjecttreenode (page

		721) object).
parent_contentobject	object	The object (as ezcontentobject (page 712) object) that is referenced by the parent node.
node	object	The actual node (as ezcontentobjecttreenode (page 721)) that this assignment assigns to the object.
temp_node	object	A temporary node (as ezcontentobjecttreenode (page 721) object) for the object that the version belongs to. The temporary node does not exist in the database.

5.5.28 ezoption

Summary

Contains information about a collection of options.

Attributes

Attribute	Type	Description
name	string	The name of the option (for example "Color").
option_list	array	An array of hashes where each hash consists of the following elements: Attribute: id Type: integer Description: The ID number of the option. Attribute: value Type: string Description: The option text (for example "Red", "Green", etc.). Attribute: additional_price Type: string Description: The addition price. Attribute: is_default Type: boolean Description: TRUE if it is the default option, FALSE otherwise.

5.5.29 ezorder

Summary

Contains information about an order.

Attributes

Attribute	Type	Description
id	string	The ID number of the order.
order_nr	string	The number (count) of the order.
is_temporary	string	1 if the order is temporary, 0 otherwise.
user_id	string	The ID number of the user that the order belongs to.
productcollection_id	string	The ID number of the product collection that belongs to the order.
data_text_1	string	General purpose text block for the shop account handler. The default shop handler uses this attribute to store an XML structure with the customer information (name, address, etc.). Example: Example: <pre><customer> <name>John Doe</name> <address>123 Main Street</address> <city>Anytown</city> <state>CA</state> <zip>12345</zip> </customer></pre>

		<pre><xml version="1.0" encoding="UTF-8"?> <shop_account> <first-name>Marty</ first-name> <last-name>McFly</ last-name> <email>marty@ez.no</ email> <street1>7 Lyon Estates</ street1> <street2>Outside Hill Valley</ street2> <zip>55532</zip> <place>Hill Valley</ place> <state>California</ state> <country>USA</country> <comment>No comment.</ comment> </shop_account></pre>
data_text_2	string	General purpose text block #2 for the shop account handler. Similar to "data_text_1".
account_identifier	string	The account identifier (for example "ez").
created	string	A UNIX timestamp pinpointing the exact date/time when the order was created.
ignore_vat	string	1 if the VAT should be ignored, 0 otherwise.
email	string	The E-mail address of the buyer.
status_id	string	The global ID number of the order's current status.
status_modified	string	A UNIX timestamp pinpointing the exact date/time when the status was order's status was last modified.
status_modifier_id	string	The ID number of the user who last modified the order's status.
status_name	string	The name of the order's

		current status (for example "Pending").
status	object	The actual order status (as an ezorderstatus (page 754) object).
status_modification_list	array	The status log as an array of ezorderstatus (page 754) objects.

5.5.30 ezorderstatus

Summary

Contains information about an order status.

Attributes

Attribute	Type	Description
id	string	The ID number of the status.
status_id	string	The global ID number of the status.
name	string	The name of the status (for example "Delivered").
is_active	string	1 if the status is active, 0 otherwise.
is_internal	boolean	TRUE if the status is one of the built-in/internal statuses, FALSE if it is a custom status.

5.5.31 ezpolicy

Summary

Contains information about a policy.

Attributes

Attribute	Type	Description
id	string	The ID number of the policy.
role_id	string	The ID number of the role the policy belongs to.
module_name	string	The name of the module that the policy grants access to (for example "content").
function_name	string	The name of the function that the policy grants access to (for example "read").
limitations	array	The limitations (if any) of the policy.
role	object	The role (as ezrole (page 759) object) that the policy belongs to.

5.5.32 ezprice

Summary

Contains information about a price.

Attributes

Attribute	Type	Description
vat_type	array	The available VAT types (as an array of ezvattype objects).
current_user	object	The current user (as an ezuser object).
is_vat_included	boolean	TRUE if the VAT is included, FALSE otherwise.
vat_percent	string	The VAT percentage (without the percent symbol).
discount_percent	double	The discount percentage.
has_discount	boolean	TRUE if a discount rule affects the price, FALSE otherwise.
selected_vat_type	object	The selected VAT type (as an ezvattype object).
price	string	The entered price.
inc_vat_price	string	The price including the VAT.
ex_vat_price	double	The price excluding the VAT.
discount_price_inc_vat	double	The discounted price including the VAT.
discount_price_ex_vat	double	The discounted price excluding the VAT.

5.5.33 ezproductcollectionitem

Summary

Contains information about an item of a product collection.

Attributes

Attribute	Type	Description
id	string	Auto-incremented ID number (used on the database level).
productcollection_id	string	The ID number of the product collection.
contentobject_id	string	The ID number of the content object.
item_count	string	The quantity.
price	string	The price of the product (the object).
is_vat_inc	string	1 if the price includes the VAT, 0 if not.
vat_value	string	The VAT value.
discount	string	Discount percentage.
name	string	The name of the product (the object).
contentobject	object	The actual content object (as an ezcontentobject (page 712) object).
option_list	array	An array representing the selected product options.

5.5.34 ezrangeoption

Summary

Contains information about a range of options.

Attributes

Attribute	Type	Description
name	string	The name of the range option (for example "Shoe size").
start_value	string	The start value (for example "32").
stop_value	string	The stop value (for example "40").
step_value	string	The step value (for example "1").
option_list	array	<p>The generated options as an array of hashes where each hash consists of the following elements:</p> <p>Attribute: id Type: integer Description: The ID of the option.</p> <p>Attribute: value Type: string Description: The option value.</p> <p>Attribute: additional_price Type: integer Description: The additional price (will always be 0).</p> <p>Attribute: is_default Type: boolean Description: TRUE if it is the default option, FALSE otherwise.</p>

5.5.35 ezrole

Summary

Contains information about a role.

Attributes

Attribute	Type	Description
id	string	The ID number of the role.
version	integer	The current version of the role, 0 is the currently active role while any other values are temporary versions.
name	string	The name of the role (for example "Anonymous").
is new	integer	The creation state of the role, will be 1 if the role just got created but have not been activated yet, 0 otherwise.
policies	array	The policies (as ezpolicy (page 755) objects) that make up the role.
limit_identifier	string	The identifier of the limited assignment for the currently logged in user. Will be FALSE if the limited assignment is not used.
limit_value	string	The value for the "limit_identifier" attribute (when the limitation feature is in use).

5.5.36 ezsection

Summary

Contains information about a section.

Attributes

Attribute	Type	Description
id	string	The ID number of the section.
name	string	The name of the section (for example "Standard").
navigation_part_identifier	string	The identifier of the navigation part that the section belongs to (for example "ezcontentnavigationpart").
locale	string	Not in use.

5.5.37 ezsimplesimplifiedxmlinput

Summary

Contains information about XML data.

Attributes

Attribute	Type	Description
input_xml	string	The text that the end-user has input to the system.
edit_template_name	string	The name of the template that is used when the object attribute is being edited. The default is "ezxmltext", but can be overridden for a custom handler (for example the Online Editor).
information_template_name	string	The name of the template that will be used when the object attribute is collecting information. The default is "ezxmltext", but can be overridden for a custom handler (for example the Online Editor).
aliased_type	string	Returns the name of the original handler. This will normally be FALSE (no alias) - the Online Editor takes control of the XML field using an alias.
alias_handler	string	Returns the original handler if the "aliased_type" attribute is non-false.

5.5.38 ezsubtreenotificationrule

Summary

Contains information about a subtree notification rule.

Attributes

Attribute	Type	Description
id	string	The ID number of the subtree notification rule.
user_id	string	The ID number of the user that the rule belongs to.
use_digest	string	When this attribute is "0", the system will check if the user has digest settings set; otherwise the digest settings are ignored and the notification is sent immediately.
node_id	string	The ID number of the subscribed node.
node	object	The actual node (as an ezcontentobjecttreenode (page 721) object).

5.5.39 eztime

Summary

Contains information about a time value.

Attributes

Attribute	Type	Description
timestamp	integer	The time as a UNIX timestamp or NULL.
hour	string	The hour.
minute	string	The minute.
is_valid	boolean	TRUE if the time is valid, FALSE otherwise.

5.5.40 ezurl

Summary

Contains information about a URL.

Attributes

Attribute	Type	Description
id	string	The ID number of the URL.
url	string	The actual address (for example "http://www.slashdot.org").
original_url_md5	string	The MD5 sum of the URL.
is_valid	string	Either 1 (valid) or 0 (invalid). Reveals if the URL is valid or not.
last_checked	string	A UNIX timestamp revealing when the URL was validated by the system.
created	string	A UNIX timestamp pinpointing the exact date/time when the URL was created.
modified	string	A UNIX timestamp pinpointing the exact date/time when the URL last modified.

5.5.41 ezuser

Summary

Contains information about a user.

Attributes

Attribute	Type	Description
contentobject_id	string	The ID number of the object that represents the user.
login	string	The username of the user.
email	string	The E-mail address of the user (for example "marty@ez.no").
password_hash	string	The encrypted version of the user's password.
password_hash_type	string	The type of encryption that was used to obfuscate the user's password.
contentobject	object	The actual object (as ezcontentobject (page 712)) that represents the user.
groups	array	The object ID numbers of the user groups that the user is a member of.
has_stored_login	boolean	Returns TRUE if the user has a non-empty username stored in the database; otherwise FALSE will be returned.
original_password	string	The password input by the user from the last page (from "/user/register" or "/content/edit"). It is only used for validation of the password. It will be FALSE if empty input was provided.
original_password_confirm	string	The confirmation password for the "original_password" attribute (FALSE if empty).
roles	array	The roles (as ezrole (page 759) objects) that are assigned to the user.
role_id_list	array	The ID numbers of the roles

		that are assigned to the user.
is_logged_in	boolean	Returns TRUE if the user is logged in, FALSE otherwise.
is_enabled	boolean	Returns TRUE if the user is enabled, FALSE otherwise.

5.5.42 ezvattype

Summary

Contains information about a VAT.

Attributes

Attribute	Type	Description
id	string	The ID number of the VAT.
name	string	The name of the VAT (for example "Standard").
percentage	string	The actual VAT percentage value (without the percent symbol).

5.5.43 ezxhtmlxmloutput

Summary

Contains information about XML data.

Attributes

Attribute	Type	Description
output_text	string	The resulting text which is suitable for the given output format, for example the rendered XHTML text.
view_template_name	string	The name of the template that will be used when the object attribute is viewed. The default is "ezxmltext", but can be overridden for a handler (for example the Online Editor).
aliased_type	string	Returns the name of the original handler. This will normally be FALSE (no alias) - the Online Editor takes control of the XML field using an alias.
aliased_handler	string	Returns the original handler if the "aliased_type" attribute is non-false.

5.5.44 ezxmlinphandler

Summary

Contains information about XML data.

Attributes

Attribute	Type	Description
input_xml	string	The text that the end-user has input to the system.
edit_template_name	string	The name of the template that is used when the object attribute is being edited. The default is "ezxmltext", but can be overridden for a custom handler (for example the Online Editor).
information_template_name	string	The name of the template that will be used when the object attribute is collecting information. The default is "ezxmltext", but can be overridden for a custom handler (for example the Online Editor).
aliased_type	string	Returns the name of the original handler. This will normally be FALSE (no alias) - the Online Editor takes control of the XML field using an alias.
alias_handler	string	Returns the original handler if the "aliased_type" attribute is non-false.

5.5.45 ezxmloutputhandler

Summary

Contains information about XML data.

Attributes

Attribute	Type	Description
output_text	string	The resulting text which is suitable for the given output format, for example the rendered XHTML text.
view_template_name	string	The name of the template that will be used when the object attribute is viewed. The default is "ezxmltext", but can be overridden for a handler (for example the Online Editor).
aliased_type	string	Returns the name of the original handler. This will normally be FALSE (no alias) - the Online Editor takes control of the XML field using an alias.
aliased_handler	string	Returns the original handler if the "aliased_type" attribute is non-false.

5.5.46 ezxmltext

Summary

Contains information about an XML block.

Attributes

Attribute	Type	Description
input	object	Returns the current input handler, which will be an object of type ezxmlinphandler (page 769), default is ezsimplifiedxmlinput (page 761).
output	object	Returns the current output handler, which will be an object of type ezxmloutphandler (page 770), default is ezxhtmloutput (page 768).
pdf_output	object	Returns the PDF output handler.
xml_data	string	Returns the internal XML structure as text.
is_empty	boolean	Returns TRUE if there is no XML data, FALSE otherwise.

5.6 Workflow events

Approve (page 773)

Makes it possible to have the contents of objects be approved by an editor.

Multiplexer (page 775)

Starts other workflows from within a workflow.

Payment gateway (page 776)

Generic solution for handling payment redirections.

Simple shipping (page 778)

Adds shipping costs to orders.

Wait until date (page 779)

Makes it possible to delay the publishing of objects.

5.6.1 Approve

Summary

Makes it possible to have the contents of objects be approved by an editor.

Description

This event makes it possible to have the contents of objects be approved by an editor before they are actually published. A workflow using this event must be connected to the "content/publish/before" trigger. The following screenshot shows the edit interface for this event.

(see figure 5.64)



Figure 5.64: Edit interface for the "Approve" event.

Affected sections

The "Affected sections" menu shows the sections (page 136) that are available on the system. It allows the administrator to filter objects that belong to specific sections. Only objects that belong to the selected sections will be affected by the event.

User who approves content

The "User who approves content" menu shows a list of the user accounts on the system. It allows the administrator to select which user(s) that should function as editors and thus have the power to approve or deny content.

Excluded users and groups

The "Excluded users and groups" menu allows the administrator to select users and user groups that should not be affected by the event. In other words, if one of the selected users or a user who is a member of a selected group publishes an object that normally would be affected by this event, it will not be affected.

This event makes use of the collaboration feature of eZ publish. A user's pending items (items waiting for approval) will be displayed under "Pending items" within the "My account" part of the administration interface. The "Collaboration" interface under "My account" provides the review/approve/reject interface for the editors.

Please note that the "cronjobs/workflow.php" cronjob must be run periodically for this to work.

5.6.2 Multiplexer

Summary

Starts other workflows from within a workflow.

Description

This event makes it possible to spawn other workflows from within a workflow. It is compatible with all of the predefined triggers. The following screenshot shows the edit interface of this event.

(see figure 5.65)

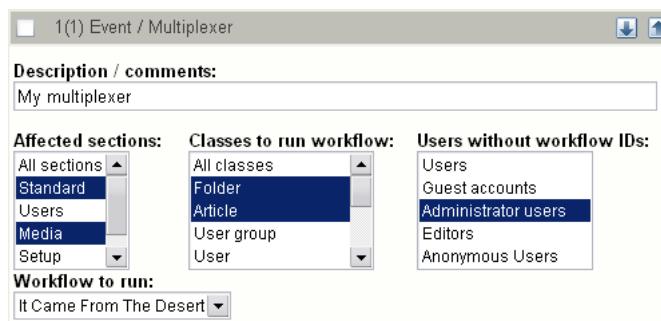


Figure 5.65: Edit interface for the "Multiplexer" event.

The "Affected sections" menu makes it possible to isolate the workflow so that it only affects content objects that belongs to the selected section(s). The "Classes to run workflow" allows a similar isolation on the class basis. In other words, only instances of the selected classes will be affected. The "Users without workflow IDs" makes it possible to select user groups that should not be affected by the multiplexer. In other words, if the current user (who initiated the workflow) is within one of the selected groups, the multiplexer event will simply be skipped by the system. The "Workflow to run" menu shows a list of the available workflows and thus allows the selection of the workflow that should be spawned by this event.

5.6.3 Payment gateway

Summary

Generic solution for handling payment redirections.

Description

This event is a general solution that is capable of handling different kinds of payment redirections. It allows several payment gateway mechanisms to be plugged into the system through a workflow. The workflow must be assigned to the "shop/checkout/before" trigger. The following screenshot shows the edit interface for this event.

(see figure 5.66)

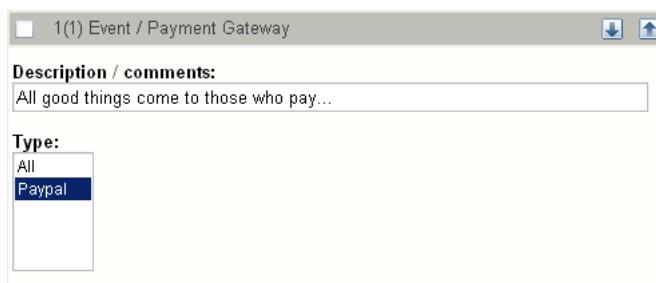


Figure 5.66: Edit interface for the "Payment gateway" event.

Type

The "Type" menu shows the payment gateway solutions that are available (installed and activated) as eZ publish extensions. It allows the administrator to select which payment gateway(s) that should be used. If several gateways are selected, the system will allow the customer to choose between the available gateways during the checkout process. The selection interface is presented using "/templates/workflow/selectgateway.tpl" located in the current or one of the fallback designs.

Execution

The following list shows the flow of execution when a payment gateway is used through the interface that this workflow event provides.

1. The customer initiates the checkout process.
2. The browser is redirected to the target payment server (PayPal, PayNet, etc.).

3. The customer attempts to pay for the products using the payment gateway.
4. The browser is sent back to eZ publish, the order will be either approved or rejected.

Between steps 3 and 4, the selected payment server will notify eZ publish about the purchase. The system will validate the authenticity (identity) of the message, and based on the reply it will either approve or reject the order. In case of no reply, the validation process will time-out and the customer will be asked to contact the shop owner.

This event provides a development framework which simplifies the implementation of payment solutions.

5.6.4 Simple shipping

Summary

Adds shipping costs to orders.

Description

This event adds shipping costs to an order (the order that is being processed for the current user). A workflow using this event must be assigned to the "shop/confirmorder/before" trigger. The following screenshot shows the edit interface for this event.

(see figure 5.67)

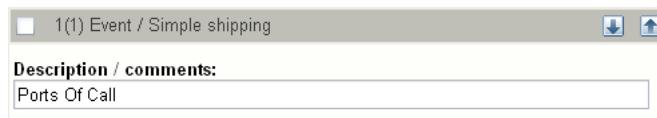


Figure 5.67: Edit interface for the "Simple shipping" event.

The cost and the description can be set using the "ShippingCost" and "ShippingDescription" directives within the "[SimpleShippingWorkflow]" group of a configuration override for "workflow.ini".

5.6.5 Wait until date

Summary

Makes it possible to delay the publishing of objects.

Description

This event makes it possible to delay the publishing of certain objects. It can be connected to either the "content/publish/before" trigger or the "content/publish/after" trigger. The following screenshot shows the edit interface for this event.

(see figure 5.68)

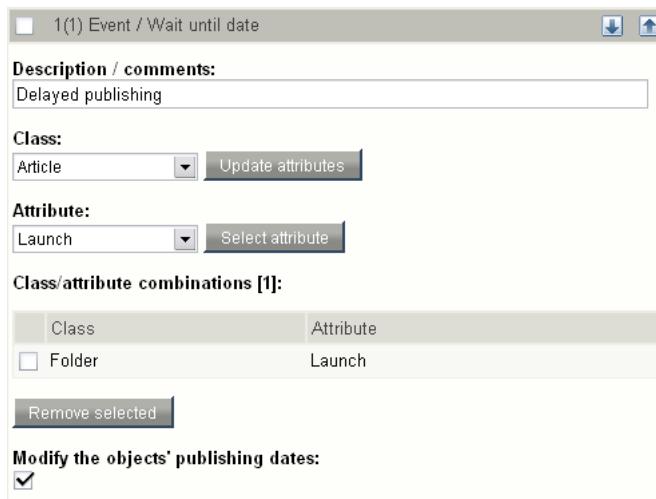


Figure 5.68: Edit interface for the "Wait until date" event.

Parameters and usage

The "Class" menu makes it possible to select a class that should be affected by the event. When a class is selected, the "Update attributes" button must be used to update the contents of the "Attribute" menu. It will allow the selection of the attribute which is used to enter the date and time when the object should be published. This attribute must be represented by the "Date and time" (page 276) datatype. Once the correct attribute is selected, the "Select attribute" button must be used to add it to the list below (labeled "Class / attribute combinations"). All objects that are instances of the classes added to this list will be affected by the event. The objects will be published automatically (by the way of the cronjob script) when the given date / time is reached.

If the "Modify the objects' publishing dates" checkbox is checked, the system will update the modification time of the objects when they are published by the system. If unchecked, the date / time when the objects were published by the users will be used.

5.7 Template operators

The template operators are documented in the following sections:

- Arrays (page [781](#))
- Data and information extraction (page [805](#))
- Formatting and internationalization (page [815](#))
- Images (page [827](#))
- Logical operations (page [837](#))
- Mathematics (page [865](#))
- Miscellaneous (page [890](#))
- Strings (page [916](#))
- URLs (page [960](#))
- Variable and type handling (page [968](#))

5.7.1 Arrays

append (page 783)

Returns the input array with appended elements.

array (page 784)

Creates and returns a new array.

array_sum (page 785)

Returns the sum of all elements in an array.

begins_with (page 786)

Checks if an array starts with a specific element/sequence.

compare (page 787)

Compares the contents of two arrays.

contains (page 789)

Checks if an array contains a specific element.

ends_with (page 791)

Checks if an array ends with a specific element or sequence.

explode (page 792)

Splits the input array and returns it as an array of sub-arrays.

extract (page 793)

Returns a portion of the input array.

extract_left (page 794)

Returns a portion of the start of the input array.

extract_right (page 795)

Returns a portion of the end of the input array.

hash (page 796)

Creates and returns a new associative array (a hash).

implode (page 797)

Joins array elements with strings.

insert (page 798)

Inserts an element/sequence at specified position in an array.

merge (page 799)

Merges input and passed arrays into one array.

prepend (page 800)

Returns the input array prepended with specified elements.

remove (page 801)

Returns the input array without some of the original elements.

repeat (page 802)

Returns a repeated version of the input array.

reverse (page 803)

Returns a reversed version of the input array.

unique (page 804)

Returns the input array without duplicate elements.

append

Summary

Returns the input array with appended elements.

Usage

```
input|append( element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
element1	any	Element to be appended to the input array.	Yes.
element2	any	Another element to be appended to the input array.	No.

Returns

An array consisting of the input array and the parameters.

Description

This operator appends the parameter value(s) at the end of the input array and returns the resulting array.

Examples

Example 1

```
{array( 1, 2, 3 )|append( 4, 5, 6 )}
```

The following array will be returned: (1, 2, 3, 4, 5, 6).

Example 2

```
{array( 1, 2, 3 )|append( array( 4, 5, 6 ) )}
```

The following array will be returned: (1, 2, 3, (4, 5, 6)).

array**Summary**

Creates and returns a new array.

Usage

```
array( element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
element1	any	Element / value of any kind.	Yes.
element2	any	Another element / value of any kind.	No.

Returns

An array containing the specified elements.

Description

This operator builds an array using the specified elements. The elements must be passed as parameters. The operator returns the resulting array.

Examples**Example 1**

```
{array( 1, 2, 3, 4, 5, 6, 7 )}
```

The following array will be returned: (1, 2, 3, 4, 5, 6, 7).

Example 2

```
{array( 1, 2, 3, array( 4, 5, 6 ) )}
```

The following array will be returned: (1, 2, 3, (4, 5, 6)).

array_sum**Summary**

Returns the sum of all elements in an array.

Usage

```
input|array_sum()
```

Returns

An integer representing the sum of the elements.

Description

This operator attempts to calculate and return the sum of the input array's elements.

Examples**Example 1**

```
{array( 1, 2, 3, 4, 5, 6, 7 )|array_sum}
```

The following output will be produced: "28" - which is the sum of 1, 2, 3, 4, 5, 6, 7.

begins_with

Summary

Checks if an array starts with a specific element/sequence.

Usage

```
input|begins_with( element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
element1	any	An element that should be matched.	Yes.
element2	any	Another element that should be matched.	No.

Returns

TRUE or FALSE.

Description

This operator checks if the input array starts with a specified sequence of elements. If yes, the operator returns TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5, 6, 7 )|begins_with( 1, 2, 3 )}
```

Returns TRUE.

Example 2

```
{array( 1, 2, 3, 4, 5, 6, 7 )|begins_with( 2, 3, 4 )}
```

Returns FALSE.

compare**Summary**

Compares the contents of two arrays.

Usage

```
input|compare( array )
```

Parameters

Name	Type	Description	Required
array	array	The array that should be compared with the input array.	Yes.

Returns

TRUE if arrays are equal, FALSE if not.

Description

This operator compares the contents of two arrays, the input array and an array that is provided as the first (and only) parameter. If the arrays are equal, the operator returns TRUE, otherwise FALSE will be returned.

Examples**Example 1**

```
{array( 1, 2, 3, 4, 5 )|compare( arrray( 1, 2, 3, 4, 5 ))}
```

Returns TRUE.

Example 2

```
{array( 1, 2, 3, 4, 5 )|compare( arrray( 5, 4, 3, 2, 1 ))}
```

Returns TRUE.

Example 3

```
{array( 1, 2, 3, 4, 5 )|compare( arrray( 1, 2, 4, 3, 3 ))}
```

Returns FALSE.

contains

Summary

Checks if an array contains a specific element.

Usage

```
input|contains( element )
```

Parameters

Name	Type	Description	Required
element	any	The element that should be matched.	Yes.

Returns

TRUE if the element is found, FALSE if not.

Description

This operator checks if the input array contains a specific element (specified using the first parameter). If it does, the operator will return TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5 )|contains( 3 )}
```

Returns TRUE.

Example 2

```
{array( 1, array( 3, 4 ), 5 )|contains( array( 3, 4 ) )}
```

Returns TRUE.

Example 3

```
{array( 1, array( 3, 4 ), 5 )|contains( 3 )}
```

Returns FALSE.

ends_with

Summary

Checks if an array ends with a specific element or sequence.

Usage

```
input|ends_with( element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
element1	any	An element that should be matched.	Yes.
element2	any	Another element that should be matched.	No.

Returns

TRUE or FALSE.

Description

This operator checks if the input array ends with a specified sequence of elements. If yes, the operator returns TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5, 6, 7 )|ends_with( 5, 6, 7 )}
```

Returns TRUE.

Example 2

```
{array( 1, 2, 3, 4, 5, 6, 7 )|ends_with( 4, 5, 6 )}
```

Returns FALSE.

explode

Summary

Splits the input array and returns it as an array of sub-arrays.

Usage

```
input|explode( offset )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset where the array should be split.	Yes.

Returns

An array containing the original array as two arrays.

Description

This operator splits the input array at an offset specified by the "offset" parameter. The operator will return an array containing the two arrays.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5 )|explode( 3 )}
```

The following array will be returned: ((1, 2, 3), (4, 5)).

extract**Summary**

Returns a portion of the input array.

Usage

```
input|extract( offset [, length ] )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	Yes.
length	integer	The number of elements that should be extracted.	No.

Returns

An array containing the extracted elements.

Description

This operator will return a portion of the input array. The desired portion must be defined by the "offset" and "length" parameters. If the "length" parameter is omitted, the rest of the array (from offset) will be returned.

Examples**Example 1**

```
{array( 1, 2, 3, 4, 5, 6, 7 )|extract( 2 )}
```

The following array will be returned: (3, 4, 5, 6, 7).

Example 2

```
{array( 1, 2, 3, 4, 5, 6, 7 )|extract( 3, 3 )}
```

The following array will be returned: (4, 5, 6).

extract_left

Summary

Returns a portion of the start of the input array.

Usage

```
input|extract_left( length )
```

Parameters

Name	Type	Description	Required
length	integer	The number of elements that should be extracted.	Yes.

Returns

An array containing the extracted elements.

Description

This operator extracts a portion from the start of the input array. The "length" parameter must be used to specify the number of elements that should be extracted.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5, 6, 7 )|extract_left( 3 )}
```

The following array will be returned: (1, 2, 3).

extract_right

Summary

Returns a portion of the end of the input array.

Usage

```
input|extract_right( length )
```

Parameters

Name	Type	Description	Required
length	integer	The number of elements that should be extracted.	Yes.

Returns

An array containing the extracted elements.

Description

This operator extracts a portion from the end of the input array. The "length" parameter must be used to specify the number of elements that should be extracted.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5, 6, 7 )|extract_right( 3 )}
```

The following array will be returned: (5, 6, 7).

hash

Summary

Creates and returns a new associative array (a hash).

Usage

```
hash( key1, value1 [, key2, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
key1	string	The key of value1.	Yes.
value1	any	The value associated with key1.	Yes.
key2	string	The key of value2.	No.
value2	any	The value associated with key2.	No.

Returns

An associative array (a hash).

Description

This operator builds an associative array using the specified key/value pairs. Odd parameters are considered to be keys, even parameters will be values. The operator returns the generated hash.

Examples

Example 1

```
{hash( 1, 'Red Eyes', 2, 'Green Gremlins', 3, 'Blue Thunder' )}
```

The following hash will be returned:

Key	Value
1	Red Eyes
2	Green Gremlins
3	Blue Thunder

implode

Summary

Joins array elements with strings.

Usage

```
input|implode( separator )
```

Parameters

Name	Type	Description	Required
separator	string	The string that should be inserted between the elements.	Yes.

Returns

String containing array elements separated a string.

Description

This operator returns a string representation of the elements of the input array. Each element will be separated by the string specified using the "separator" parameter.

Examples

Example 1

```
{array( 1, 2, 3, 4, 5, 6, 7 )|implode( ', ' )}
```

The following string will be returned: "1, 2, 3, 4, 5, 6, 7".

Example 2

```
{array( 1, 2, 3, 4, 5, 6, 7 )|implode( '_-' )}
```

The following string will be returned: "1_-2_-3_-4_-5_-6_-7".

insert

Summary

Inserts an element/sequence at specified position in an array.

Usage

```
input|insert( offset, element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset where the element(s) should be inserted at.	Yes.
element1	any	An element that should be inserted into the existing array.	Yes.
element2	any	Another element that should be inserted into the existing array.	No.

Returns

An array containing a combination of the original array and the inserted elements.

Description

This operator inserts an element or a sequence of elements at a specified position within the input. The resulting array will be returned (original array with the inserted values).

Examples

Example 1

```
{array( 1, 2, 5 )|insert( 2, 3, 4 )}
```

The following array will be returned: (1, 2, 3, 4, 5).

merge**Summary**

Merges input and passed arrays into one array.

Usage

```
input|merge( array1 [, array2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
array1	array	Array to be merged with the input array.	Yes.
array2	array	Another array to be merge with the input array.	No.

Returns

New array containing all arrays merged.

Description

This operator will merge the input array with all arrays passed as parameters. The resulting array will be returned.

Examples**Example 1**

```
{array( 1, 2 )|merge( array( 3, 4 ), array( 5, 6, 7 ) )}
```

The following array will be returned: (1, 2, 3, 4, 5, 6, 7).

prepend

Summary

Returns the input array prepended with specified elements.

Usage

```
input|prepend( element1 [, element2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
element1	any	Element to be prepended to the input array.	Yes.
element2	any	Another element to be prepended to the input array.	No.

Returns

An array consisting of the parameters and the input array.

Description

This operator adds the parameter value(s) to the start of the input array and returns the resulting array.

Examples

Example 1

```
{array( 4, 5 )|prepend( 1, 2, 3 )}
```

The following array will be returned: (1, 2, 3, 4, 5).

remove**Summary**

Returns the input array without some of the original elements.

Usage

```
input|remove( offset [, length ] )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start removing elements.	Yes.
length	integer	The number of elements that should be removed.	No.

Returns

A cut-down version of the input array.

Description

This operator removes element(s) from the input array and thus a chopped/cut-down version of the input array will be returned. The "offset" parameter must be used to define the start of the portion that should be removed. The "length" parameter can be used to define the number of elements that should be removed. If the "length" parameter is omitted, only one element (specified by offset will be removed).

Examples**Example 1**

```
{array( 1, 2, 3, 4, 5 )|remove( 2, 2 )}
```

The following array will be returned: (1, 2, 5).

Example 2

```
{array( 1, 2, 3, 4, 5 )|remove( 2 )}
```

The following array will be returned: (1, 2, 4, 5).

repeat**Summary**

Returns a repeated version of the input array.

Usage

```
input|repeat( times )
```

Parameters

Name	Type	Description	Required
times	integer	The number of times the array should be repeated.	Yes.

Returns

An array containing the elements of the input array repeated n times.

Description

This operator returns a repeated version of the input array. The "times" parameter must be used to define the number of times the array should be repeated.

Examples**Example 1**

```
{array( 1, 2, 3, 4 )|repeat( 3 )}
```

The following array will be returned: (1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4).

reverse**Summary**

Returns a reversed version of the input array.

Usage

```
input|reverse()
```

Returns

A reversed version of the input array.

Description

This operator returns a reversed version (elements in reverse order) of the input array.

Examples**Example 1**

```
{array( 1, 2, 3, 4 )|reverse}
```

The following array will be returned: (4, 3, 2, 1).

unique**Summary**

Returns the input array without duplicate elements.

Usage

```
input|unique()
```

Returns

The input array containing only one occurrence of every element.

Description

This operator removes duplicate elements from the input array.

Examples**Example 1**

```
{array( 1, 2, 2, 3, 4, 4, 5 )|unique}
```

The following array will be returned: (1, 2, 3, 4, 5).

5.7.2 Data and information extraction

currentdate (page 806)

Returns the timestamp of the current date/time.

ezhttp (page 807)

Returns GET, POST and session variables.

ezini (page 809)

Provides read access to settings in the configuration files.

ezpreference (page 810)

Provides access to a user's preference values.

ezsys (page 811)

Returns misc values such as wwwdir, sitedir, etc.

fetch (page 813)

Provides access to the fetch functions of a module.

module params (page 814)

Extracts parameters from the module that was run.

currentdate

Summary

Returns the timestamp of the current date/time.

Usage

```
currentdate()
```

Returns

The current date/time as a UNIX timestamp.

Description

This operator returns the UNIX timestamp for the current date/time.

Examples

Example 1

```
{def $timestamp=currentdate()  
Current timestamp: {$timestamp}}
```

Outputs the current timestamp.

Example 2

```
{def $timestamp=currentdate()  
Current date/time: {$timestamp|l10n( 'shortdatetime' )}}
```

Outputs the current date/time in a user friendly format.

ezhttp

Summary

Returns GET, POST and session variables.

Usage

```
ezhttp( name [, type ] )
```

Parameters

Name	Type	Description	Required
name	string	The HTTP variable that should be fetched.	Yes.
type	string	Source of variable. Default is is HTTP POST.	No.

Returns

Variable value

Description

This operator makes it possible to inspect the contents of HTTP variables (POST, GET or session variables). The "type" parameter can be used to specify which type of variable that should be extracted. The following options are available:

- post (POST variable)
- get (GET variable)
- session (session variable)

Examples

Example 1

```
{ezhttp( 'search' )}
```

Returns the "search" POST variable.

Example 2

```
{ezhttp( 'image', 'get' )}
```

Returns the "image" GET variable.

Example 3

```
{ezhttp( 'user_id', 'session' )}
```

Returns the "user_id" session variable.

ezini

Summary

Provides read access to settings in the configuration files.

Usage

```
ezini( section, variable [, ini file] )
```

Parameters

Name	Type	Description	Required
section	string	Section to read value from.	Yes.
variable	string	The name of the directive that should be accessed.	Yes.
ini file	string	The target configuration file (default is "site.ini").	No.

Returns

A string containing the value of a configuration setting.

Description

This operator makes it possible to access the settings in the configuration ("*.ini") files.

Examples

Example 1

```
{if eq(ezini( 'SomeSettings', 'Test', 'example.ini' ), 'hello' )}
    [...display something...]
{else}
    [...display something else...]
{/if}
```

Conditional branching based on an configuration setting.

ezpreference

Summary

Provides access to a user's preference values.

Usage

```
ezpreference( preference )
```

Parameters

Name	Type	Description	Required
preference	string	The name of the preference that should be extracted.	Yes.

Returns

A string containing the contents/value of the specified preference.

Description

This operator makes it possible to extract the preference values of the current user. The name of the desired preference must be provided as a parameter. The function returns the value/contents of the specified preference. The current user can set a preference by requesting a URL of the following type:

```
/user/preferences/set/[name_of_preference]/[value]
```

Examples

Example 1

```
{ezpreference( 'bookmark_menu' )}
```

Returns the value/contents of the "bookmark_menu" preference for the current user.

ezsys

Summary

Returns misc values such as wwwdir, sitedir, etc.

Usage

```
ezsys( system_variable )
```

Parameters

Name	Type	Description	Required
system_variable	string	The name of the desired variable.	Yes.

Returns

A string containing the requested variable.

Description

This operator gives read access to certain eZ Publish system variables. The "system_variable" parameter must be used to specify the name of the variable that should be returned. The following names can be used:

- indexfile - name of index file
- indexdir - relative path and index file
- sitedir - local path of eZ Publish installation
- wwwdir - relative directory path of eZ Publish installation
- hostname - hostname of eZ Publish server

Examples

Example 1

Example: news theme installed on server "ez.no", by user "tom" in "/home/tom/public_html/local/ez_publish"

```
wwwdir - {ezsys( 'wwwdir' )}<br />
sitedir - {ezsys( 'sitedir' )}<br />
indexfile - {ezsys( 'indexfile' )}<br />
indexdir - {ezsys( 'indexdir' )}<br />
magicQuotes - {ezsys( 'magicQuotes' )}<br />
hostname - {ezsys( 'hostname' )}<br />
```

The following output will be produced:

```
wwwdir - ~/tom/local/ez_publish
sitedir - /home/tom/public_html/local/ez_publish/
indexfile - /index.php/news
indexdir - ~/tom/local/ez_publish/index.php/news
magicQuotes -
hostname - ez.no
```

fetch

Summary

Provides access to the fetch functions of a module.

Usage

```
fetch( ... )
```

Parameters

Name	Type	Description	Required
any	any	Please refer to the links below.	Yes.

Returns

The returned result depends on the actual fetch operation.

Description

The fetch operator provides access to the fetch functions of an eZ Publish module. Please refer to the "Information extraction" (page 217) section of the "Templates" chapter for more information about the operator itself. The actual fetch functions are documented under the "Template fetch functions" (page 1088) section of the "Reference" chapter.

module_params

Summary

Extracts parameters from the module that was run.

Usage

```
module_params()
```

Returns

An array containing module information.

Description

This operator extracts some generic information from the module that was run. It seems that it can only be called inside "pagelayout.tpl". Please refer to the example below. The operator does not take any parameters.

Examples

Example 1

```
{module_params()|attribute(show)}
```

If the requested URL is "/content/view/full/65" (or using URL alias that points to "/content/view/full/65"), the following output will be produced:

Attribute	Type	Value
module_name	string	'content'
function_name	string	'view'
parameters	array	Array(2)
-ViewMode	string	'full'
-NodeID	string	65

5.7.3 Formatting and internationalization

datetime (page 816)

Formats dates/times according to settings in "datetime.ini".

i18n (page 819)

Marks a string for translation.

l10n (page 821)

Formats misc. numbers (times, dates, currencies, numbers, etc.).

si (page 823)

Handles unit display of values (output formatting).

datetime

Summary

Formats dates/times according to settings in "datetime.ini".

Usage

```
{input|datetime( preset_format [, format ] )}
```

Parameters

Name	Type	Description	Required
preset_format	string	Preset datetime format set in "datetime.ini".	Yes.
format	string	Custom format (when preset format is set to "custom").	No.

Returns

A string representation of the input parameter.

Description

This operator takes care of formatting dates and times according to the setting defined in "datetime.ini" (or a configuration override). In addition, the operator also allows custom formats when the "preset_format" parameter is set to "custom". A custom format must be specified using the "format" parameter. The following table reveals the different elements that can be used in a custom format.

Element	Output	Description
%a	am	Lowercase Ante meridiem and Post meridiem.
%A	AM	Uppercase Ante meridiem and Post meridiem.
%d	08	Day of the month, 2 digits with leading zeros.
%D	Wed	A short textual representation of a day, in accordance with the "[ShortDayNames]" section of the language .INI file located in the "share/locale" directory.
%F	October	A full textual representation of a month, such as January

		or March.
%g	12	12-hour format of an hour without leading zeros.
%G	0	24-hour format of an hour without leading zeros.
%h	12	12-hour format of an hour with leading zeros.
%H	00	24-hour format of an hour with leading zeros.
%i	00	Minutes with leading zeros
%j	8	Day of the month without leading zeros
%l	Wednesday	A full textual representation of the day of the week.
%m	10	Numeric representation of a month, with leading zeros.
%M	Oct	A short textual representation of a month, in accordance with the "[ShortMonthNames]" section of the language .INI file located in the "share/locale" directory.
%n	10	Numeric representation of a month, without leading zeros.
%O	-0500	Difference to Greenwich time (GMT) in hours.
%s	00	Seconds, with leading zeros.
%T	CDT	Timezone setting of this machine.
%U	1065589200	Seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).
%w	3	Numeric representation of the day of the week.
%W	41	ISO-8601 week number of year, weeks starting on Monday.
%Y	2003	A full numeric representation of a year, 4 digits.
%y	03	A two digit representation of a year.
%z	280	The day of the year.
%Z	-18000	Timezone offset in seconds. The offset for timezones west of UTC is always negative,

and for those east of UTC is always positive.

The date used to generate the contents of this table was "12:00 AM (Midnight) CDT on October 8, 2003".

Examples

Example 1

```
{currentdate()|datetime( 'mydate' )}
```

The following output will be produced: "13:15 6 Feb 2004" (according to the configuration settings).

Example 2

```
{currentdate()|datetime( 'custom', '%h:%i %a %d %F %Y' )}
```

The following output will be produced: "01:15 pm 06 February 2004" (custom format).

i18n

Summary

Marks a string for translation.

Usage

```
input|i18n( [ context [, comment [, arguments ] ] ] )
```

Parameters

Name	Type	Description	Required
context	string	The context to which the string belongs.	No.
comment	string	A comment describing the text.	No.
arguments	hash	An associative array of arguments in the input parameter.	No.

Returns

A string containing a translated version of the input parameter.

Description

This operator makes it possible to translate static strings that are defined in various templates. It is typically useful to ensure that the HTML interface is available in several languages in a multilanguage scenario.

The operator takes three optional parameters: "context", "comment" and "arguments". The "context" parameter can be used to specify a group to which the input parameter should be related. This is typically useful when there are a lot of strings that need to be structured/grouped. The "comment" parameter makes it possible to add additional comment which can help the person responsible for doing the actual translation. A comment could for example be "Button label" - revealing what that mysterious string actually is. The "arguments" parameter makes it possible to mix dynamic text into the translations. Please refer to "Example 2" for a demonstration of this feature.

Examples

Example 1

```
{"This is a test"|i18n}
```

Outputs "This is a test" translated to the current language.

Example 2

```
{def $number=5}  
{"Please enter %number characters."|i18n( ', ', hash( '%number', $number ) )}
```

Outputs "Please enter 5 characters.", the %number will be dynamically replaced by the variable.

Example 3

```
{"Are you sure you want to remove these items?"|i18n('design/standard/node')}
```

Outputs "Are you sure you want to remove these items?" translated to the current language. The translation is taken from the context block named "design/standard/node" located in the appropriate translation file.

For example, let's say that the "translation.ts" file located in the "share/translations/ita-IT/" directory of your eZ Publish installation contains the following lines:

```
<context>  
<name>design/standard/node</name>  
...  
<message>  
<source>Are you sure you want to remove these items?</source>  
<translation>Sei sicuro di voler rimuovere questi elementi?</translation>  
</message>  
...  
</context>
```

If your current system locale is "ita-IT" (as specified in the " Locale (page 1299)" setting located in the "[RegionalSettings]" section of the "settings/site.ini" configuration file or its override) then the following output is produced: "Sei sicuro di voler rimuovere questi elementi?"

110n

Summary

Formats misc. numbers (times, dates, currencies, numbers, etc.).

Usage

```
input|110n( type )
```

Parameters

Name	Type	Description	Required
type	string	The format that should be used.	Yes.

Returns

A string containing a formatted version of the input parameter.

Description

This operator formats/localizes miscellaneous numeric values according to the current locale settings. The value that should be formatted must be input using the input parameter. The "type" parameter must be used to select the desired format. The following list reveals the available formats/types.

- time
- shorttime
- date
- shortdate
- datetime
- shortdatetime
- currency
- clean_currency
- number

Examples

Example 1

```
{def $number=1234.567
    $timestamp=currentdate()

time:      {$timestamp|l10n( 'time' )}
shorttime: {$timestamp|l10n( 'shorttime' )}
date:      {$timestamp|l10n( 'date' )}
shortdate: {$timestamp|l10n( 'shortdate' )}
datetime:   {$timestamp|l10n( 'datetime' )}
shortdatetime: {$timestamp|l10n( 'shortdatetime' )}
currency:   {$number|l10n( 'currency' )}
clean_currency: {$number|l10n( 'clean_currency' )}
number:     {$number|l10n( 'number' )}
```

If the current locale is "eng-GB", the following output will be produced:

```
time : 1:46:05 pm
shorttime : 1:46 pm
date : Friday 06 February 2004
shortdate : 06/02/2004
datetime : Friday 06 February 2004 1:46:05 pm
shortdatetime : 06/02/2004 1:46 pm
currency : £ 1,234.57
clean_currency : 1,234.57
number : 1,234.57
```

si

Summary

Handles unit display of values (output formatting).

Usage

```
input|si( unit [, prefix ] [, decimals ] [, symbol ] [, separator ] )
```

Parameters

Name	Type	Description	Required
unit	string	The unit that the input value should be converted to.	Yes.
prefix	string	The prefix used to represent the input value.	No.
decimals	integer	The number of decimal digits that should be shown.	No.
symbol	string	The symbol that should be used as the decimal separator.	No.
separator	string	The symbol that should be used as the thousand separator.	No.

Returns

The input value formatted according to the given parameters.

Description

This operator handles formatting of different kind of values (file sizes, lengths, weights, etc.). The value that should be formatted must be provided as the input parameter. The first parameter must reveal the type of the input value. The following list shows the types that can be used.

- meter
- gram
- second
- ampere
- kelvin
- mole
- candela

- byte
- bit

The second parameter is optional and can be used to specify a desired prefix. The rest of the parameters determine how the value should be formatted when it comes to decimals, separators, etc. If these parameters are omitted, the operator will use the settings of the current locale.

Custom units

Custom units can be configured by extending the entries of the "Base" group in a configuration override for "units.ini".

Prefix tables

The prefix is either the name of the size like *kilo* or one of these

Prefix	Description
binary	Calculate using 2 as base, e.g. 2^8
decimal	Calculate using 10 as base, e.g. 10^6
none	Show value as it is with just the unit appended
auto	Determine base from the type of unit (controlled by the "BinaryUnits" setting, default is "bit" and "byte").

The following table shows the proper binary prefixes.

Prefix	Power of 2	Symbol
kibi	2^{10}	Ki
mebi	2^{20}	Mi
gibi	2^{30}	Gi
tebi	2^{40}	Ti
pebi	2^{50}	Pi
exbi	2^{60}	Ei

The following table shows the commonly used binary prefixes (please note that they may be inaccurate).

Prefix	Power of 2	Symbol
kilo	2^{10}	k
mega	2^{20}	M
giga	2^{30}	G
tera	2^{40}	T
peta	2^{50}	P
exa	2^{60}	E

The following table shows the decimal prefixes.

Prefix	Power of 10	Symbol
yotta	10^{24}	Y
zetta	10^{21}	Z
exa	10^{18}	E
peta	10^{15}	P
tera	10^{12}	T
giga	10^9	G
mega	10^6	M
kilo	10^3	k
hecto	10^2	h
deca	10^1	da
deci	10^{-1}	d
centi	10^{-2}	c
milli	10^{-3}	m
micro	10^{-6}	
nano	10^{-9}	n
pico	10^{-12}	p
femto	10^{-15}	f
atto	10^{-18}	a
zepto	10^{-21}	z
yocto	10^{-24}	y

All of these values are defined in the "units.ini" configuration file.

Examples

Example 1

```
{1025|si( byte )}
{1025|si( byte, binary )}
{1025|si( byte, decimal )}
{1025|si( byte, none )}
{1025|si( byte, auto )}
{1025|si( byte, kibi )}
{1025|si( byte, kilo )}
```

If the "UseSIUnits" directive in a configuration override for "site.ini" is set to "false" (the default value), the following output will be produced:

```
1.00 kB
1.00 kB
1.02 kB
1025 B
1.00 kB
```

```
1.00 KiB  
1.02 kB
```

If the "UseSIUnits" directive in a configuration override for "site.ini" is set to "true", the following output will be produced:

```
1.00 KiB  
1.00 KiB  
1.02 kB  
1025 B  
1.00 KiB  
1.00 KiB  
1.02 kB
```

5.7.4 Images

image (page 828)

Creates and returns an image object.

imagefile (page 831)

Loads an image from a file.

texttoimage (page 832)

Renders a string as an image using a truetype font.

image

Summary

Creates and returns an image object.

Usage

```
image( value [,...] )
```

Parameters

Name	Type	Description	Required
value	mixed	Please refer to the description below.	Yes.

Returns

An ezimage object.

Description

This operator makes it possible to merge/flatten several images into one image. It returns the final image as an "ezimage" (page 735) object. Please note that this operator only works if "ImageGD" is installed and enabled. It takes several parameters where each parameter can be a different type. The following types can be used:

- String
- Image layer (an "ezimagelayer" (page 734) object).
- Array

String

If a parameter is a string, the contents of the string will be used as the alternative image text for the image object that is returned by the operator.

Image layer

If a parameter is an image layer (an "ezimagelayer" (page 734) object), it will be merged with the other layers and thus become a part of the final result. An image layer may be generated by making use of the "imagefile" (page 831) or the "texttoimage" (page 832) operator.

Array

If a parameter is an array, the operator will assume that the first element (element number zero) is an image layer and that the second element is an associative array containing parameters that the system will use when the image layer is processed. The following table shows the parameters that can be used.

Name / key	Description
transparency	The image transparency as float value ranging from 0.0 (0% transparency) to 1.0 (100% transparency).
halign	Horizontal alignment as a string: "left", "right" or "center".
valign	Vertical alignment as a string: "top", "bottom" or "center".
x	Absolute horizontal placement as an integer (works with "left" and "right" horizontal alignment.)
y	Absolute vertical placement as an integer (works with "top" and "bottom" vertical alignment.)
xrel	Relative horizontal placement as a float value ranging from 0.0 to 1.0 (works with "left" and "right" horizontal alignment.)
xrel	Relative vertical placement as a float value ranging from 0.0 to 1.0 (works with "top" and "bottom" vertical alignment.)

Please note that the "x" and "xrel" parameters can not be used at the same time. The same goes for the "y" and the "yrel" parameters. When "right" or "bottom" alignment is used, the coordinate system will shift to accommodate the alignment. This is useful for doing alignment and placement since the placement is relative to the current coordinate system. Right alignment will start the axis at the right (0) and go on to the left (width). Bottom alignment will start the axis at the bottom (0) and go on to the top (height).

Template

If the operator is called directly, eZ Publish will display the resulting image using the "layer/imageobject.tpl" template located in the "templates" directory of the current design or one of the fallback designs. It is possible to override this template using the template override system (page 222).

Examples

Example 1

```
{image( imagefile( 'design/example/images/test1.png' ),
        imagefile( 'design/example/images/test2.png' ),
        'My alternative image text...')}
```

The "imagefile" (page 831) operator loads the images from the filesystem and returns them as "ezimagerlayer" (page 734) objects. The "image" operator takes care of merging the images (layers) together into one single image. When the "image" operator is used directly (as in the example above), eZ Publish will insert the "imageobject.tpl" template (or an override) which in turn takes care of displaying the image. The last parameter is a string, which will be used as the alternative image text for the final image.

Example 2

```
{image( "It's A Kind Of Magic" | texttoimage( 'arial' ) )}
```

This example will render the string "It's A Kind Of Magic" using a truetype font specified within the "arial" style. The image layer produced by the "texttoimage" operator is wrapped inside an image object and displayed using the "imageobject.tpl" template (or an override).

Example 3

```
{image( "It's A Kind Of Magic",
        imagefile( 'design/example/images/test1.png' ),
        array( "It's A Kind Of Magic" | texttoimage( 'smartie' ),
               hash( 'transparency', 0.64,
                     'halign',           'right',
                     'valign',           'top' ) ) )}
```

This example will generate an image with 64% transparent text using the "smartie" style. The text will be aligned to the upper right corner of the "example1.png" image. The alternative image text will be "It's A Kind Of Magic" - which is the same as the merged in transparent text.

imagefile

Summary

Loads an image from a file.

Usage

```
imagefile( filename )
```

Parameters

Name	Type	Description	Required
filename	string	The image file that should be loaded.	Yes.

Returns

An ezimagelayer object.

Description

This operator loads an image from the filesystem and returns it as an "ezimagelayer" (page 734) object. The location and the name of the file must be specified in the same way as it is shown in the example below (the whole path must be specified without a starting slash). Please note that this operator will only work if "ImageGD" is installed and enabled.

If this operator is called directly, eZ Publish will display the specified image using the "image.tpl" template located in the "templates/image/" directory of the current design or one of the fallback designs. It is possible to override this template using the template override system (page 222). An image layer can be provided as a parameter to the "image" (page 828) operator.

Examples

Example 1

```
{imagefile( 'design/example/images/test.png' )}
```

This will load "test.png" from the specified directory and display it using the image layer template.

texttoimage

Summary

Renders a string as an image using a truetype font.

Usage

```
input|texttoimage( style )
```

Parameters

Name	Type	Description	Required
style	string	The name of the style that should be used.	Yes.

Returns

Image layer.

Description

This operator creates an image that contains the input text rendered using a truetype font. The style parameter must be used to specify the desired style. If the style parameter is omitted, the default style will be used. The operator returns an "ezimagelayer" (page 734) object which can be used as a parameter to the "image" (page 828) operator. If this operator is called directly, eZ Publish will display the specified image using the "layer/image.tpl" template located in the "templates" directory of the current design or one of the fallback designs. It is possible to override this template using the template override system (page 222). Please note that this operator will only work if "ImageGD" is installed and enabled.

An eZ Publish distribution comes with a small collection of truetype fonts that are used by the default styles. These fonts are located in the "/design/standard/fonts" directory. The fonts included in this directory are free when it comes to costs and distribution. Information about the author of a font is placed in a directory with the same name as the font itself.

Default styles

The default styles are defined in the "texttoimage.ini" configuration file. The following list reveals the names of the default styles.

- 1942
- a_d_mono

- archtura
- arial
- gallery
- object_text
- sketchy
- smartie

Please refer to the examples below to see the default styles in action.

Custom styles

It is possible to create custom styles and to make use of custom fonts. For each style, it is possible to configure the following settings:

- The name of the style
- Font family
- Point size
- Background color
- Text color
- Angle/rotation
- X adjustment
- Y adjustment
- Width adjustment
- Height adjustment
- Absolute width
- Absolute height

When using fonts that are located outside the "/design/standard/fonts" directory, for example "/design/example/fonts", the `FontDir[]` array in a configuration override for "texttoimage.ini" has to include an additional `FontDir[]` line that specifies the secondary font directory.

Examples

Example 1

```
{'Another World'|texttoimage( '1942' )}
```

The following output will be produced:

(see figure 5.69)

The text "Another World" is rendered in a bold, black, monospace-style font. The letters are slightly rounded and have a distinct vertical stroke.

Figure 5.69: Text rendered as image using the 1942 font.

Example 2

```
{'Another World'|texttoimage( 'a_d_mono' )}
```

The following output will be produced:

(see figure 5.77)

The text "ANOTHER WORLD" is rendered in a bold, black, monospace-style font. The letters are more rounded and have a thicker vertical stroke compared to the 1942 font.

Figure 5.70: Text rendered as image using the a_d_mono font.

Example 3

```
{'Another World'|texttoimage( 'archtura' )}
```

The following output will be produced:

(see figure 5.71)

The text "Another World" is rendered in a bold, blue, monospace-style font. The letters have a distinct vertical stroke and a slightly different character design.

Figure 5.71: Text rendered as image using the archtura font.

Example 4

```
{'Another World'|texttoimage( 'arial' )}
```

The following output will be produced:

(see figure 5.72)



Another World

Figure 5.72: Text rendered as image using the arial font.

Example 5

```
{'Another World'|texttoimage( 'gallery' )}
```

The following output will be produced:

(see figure 5.73)



Another World

Figure 5.73: Text rendered as image using the gallery font.

Example 6

```
{'Another World'|texttoimage( 'object_text' )}
```

The following output will be produced:

(see figure 5.74)



Another World

Figure 5.74: Text rendered as image using the object_text font.

Example 7

```
{'Another World'|texttoimage( 'sketchy' )}
```

The following output will be produced:

(see figure 5.75)



ANOTHER WORLD

Figure 5.75: *Text rendered as image using the sketchy font.*

Example 8

```
{'Another World'|texttoimage( 'smartie' )}
```

The following output will be produced:

(see figure 5.76)



Another World

Figure 5.76: *Text rendered as image using the smartie font.*

Example 9

```
{'Another World'|texttoimage()}
```

The following output will be produced (the default font will be used):

(see figure 5.77)



ANOTHER WORLD

Figure 5.77: *Text rendered as image using the a_d_mono font.*

5.7.5 Logical operations

and (page 838)

Evaluates all parameters to check if any are evaluated as FALSE.

choose (page 840)

Returns one of the parameters (pinpointed by the input parameter).

cond (page 841)

Returns the value of the first clause who's condition is TRUE.

eq (page 843)

Returns TRUE if the input equals the first parameter or if all parameters are equal.

false (page 846)

Creates and returns a boolean FALSE.

first_set (page 847)

Returns the first parameter that is set (or FALSE).

ge (page 849)

Returns TRUE if a parameter is greater than or equal to another parameter.

gt (page 851)

Returns TRUE if a parameter is greater than another parameter.

le (page 853)

Returns TRUE if a parameter is less than or equal to another parameter.

lt (page 855)

Returns TRUE if a parameter is less than another parameter.

ne (page 857)

Returns TRUE if one or more of the parameters do not match.

not (page 859)

Returns the opposite of the input or the first parameter (TRUE/FALSE).

null (page 861)

Returns TRUE if the input value is NULL (not the same as 0).

or (page 862)

Evaluates all parameters until one is found to be TRUE, returns that value.

true (page 864)

Creates and returns a boolean TRUE.

and

Summary

Evaluates all parameters to check if any are evaluated as FALSE.

Usage

```
and( value1 [, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value to be evaluated.	Yes.
value2	any	Another variable/value to be evaluated.	No.

Returns

One of the parameters or TRUE (see below).

Description

This operator evaluates all parameters.

When one of them is found to be FALSE, the operator will then return that parameter. Under some circumstances shortcircuiting is then applied, which means that once a parameter is found to be FALSE, the evaluation of the rest of the parameters stops. However this is not always the case, and depends on personal settings.

If none of the parameters are found to be FALSE, the operator will return TRUE.

The following table shows how the different types are evaluated by this operator.

Type	Evaluation
Number	TRUE if the value is non-zero, FALSE otherwise.
String	TRUE if the string consists of at least one character, FALSE otherwise.
Boolean	TRUE if the boolean is a TRUE value, FALSE is otherwise.
Array	TRUE if the array has one or more elements, FALSE otherwise.
Object	TRUE if the object provides the "attributes" and the "attribute" methods, FALSE otherwise.
NULL	Always FALSE.
Other	Other types will be evaluated in the same way as PHP would do.

Examples

Example 1

```
{if and( false(), true(), false() )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The day the earth stood still.".

Example 2

```
{def $a=array()  
    $b=array( 1, 2, 3 )  
    $c=array( 4, 5, 6 )}  
  
{and( $a, $b, $c )}
```

The code above will return the empty array that is represented by \$a.

choose

Summary

Returns one of the parameters (pinpointed by the input parameter).

Usage

```
input|choose( value1 [, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value of any kind.	Yes.
value2	any	Another variable/value of any kind.	No.

Returns

One of the parameters.

Description

This operator returns one of the parameters. The input parameter must be an integer that pinpoints exactly which parameter that should be returned. If the input parameter is zero, the operator will return the first parameter. If the input parameter is one, the operator will return the second parameter, and so on. If the offset is wrong, the operator will return FALSE.

Examples

Example 1

```
{0|choose( 'apples', 'bananas', 'coconuts' )}
```

The following output will be produced: "apples".

Example 2

```
{2|choose( 'apples', 'bananas', 'coconuts' )}
```

The following output will be produced: "coconuts".

cond

Summary

Returns the value of the first clause who's condition is TRUE.

Usage

```
cond( cond1, value1 [, cond2, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
cond1	boolean	Match condition 1.	Yes.
value1	any	Return value for condition 1.	Yes.
cond2	boolean	Match condition 2.	No.
value2	any	Return value for condition 2.	No.

Returns

One of the provided values or FALSE.

Description

This operator evaluates the provided conditions (odd numbered parameters). If one of the conditions evaluates to TRUE then the value which is associated with that condition is returned. If none of the conditions are TRUE, the operator will return FALSE. If an odd number of parameters are provided, the operator will return the last parameter if all conditions fail.

Examples

Example 1

```
{cond( true(), 'apples', true(), 'bananas' )}
```

The following output will be produced: "apples".

Example 2

```
{cond( false(), 'apples', true(), 'bananas' )}
```

The following output will be produced: "bananas".

Example 3

```
{cond( false(), 'apples', false(), 'bananas' )}
```

No output will be produced, the operator will return FALSE.

Example 4

```
{cond( false(), 'apples', 'bananas' )}
```

The following output will be produced: "bananas".

eq

Summary

Returns TRUE if the input equals the first parameter or if all parameters are equal.

Usage

```
input|eq( value1 [, value2 ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared.	Yes.
value2	any	Another variable/value that should be compared.	No.

Returns

TRUE or FALSE (see below).

Description

This operator compares the contents of two or more variables and/or values. If the input parameter is used, the operator will compare it with the first parameter. If the provided variables/values match, the operator will return TRUE, otherwise FALSE will be returned. If more than one parameter is provided, the operator will compare all parameters. If all parameters are found to be equal, the operator will return TRUE, otherwise FALSE will be returned. If more than one parameter is provided, the operator will simply ignore the input parameter.

Note that "eq" compares the values in the same way as the '==' operator in PHP programming language (for example, 0.1 and 0.10 will be equal). Refer to the [PHP reference documentation](#) for more information. It is recommended to use the €compare (page 923)€ template operator for string comparision.

Examples

Example 1

```
{if 1|eq( 1 )}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The truth is out there".

Example 2

```
{if 1|eq( 2 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The day the earth stood still".

Example 3

```
{if eq( 1, 1 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The truth is out there".

Example 4

```
{if eq( 1, 2 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The day the earth stood still".

Example 5

```
{if 1|eq( 1, 1 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The truth is out there".

Example 6

```
{if 2|eq( 1, 1 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The truth is out there.".

Example 7

```
{if 1|eq( 1, 2 )}  
    The truth is out there.  
{else}  
    The day the earth stood still.  
{/if}
```

The following output will be produced: "The day the earth stood still".

false

Summary

Creates and returns a boolean FALSE.

Usage

```
false()
```

Returns

FALSE.

Description

This operator creates and returns a boolean FALSE. It can be used to define a boolean variable and in logical comparisons.

Examples

Example 1

```
{def $my_boolean=false()}

{if $my_boolean}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The day the earth stood still.".

first_set

Summary

Returns the first parameter that is set (or FALSE).

Usage

```
first_set( value1 [, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be evaluated.	Yes.
value2	any	Another variable/value that should be evaluated.	No.

Returns

The first value that is set or FALSE.

Description

This operator evaluates all parameters until one of them is found to be set. The parameter that is found to be set will be returned. If none of the parameters are set, the operator will return FALSE.

Examples

Example 1

```
{if first_set( $a, $b, $c )}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

As long as \$a, \$b and \$c are undeclared/unset, the following output will be produced: "The day the earth stood still".

Example 2

```
{first_set( $a, 256, $b )}
```

As long as \$a is undeclared/unset, the following output will be produced: "256".

ge

Summary

Returns TRUE if a parameter is greater than or equal to another parameter.

Usage

```
input|ge( value1 [, value2 ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared.	Yes.
value2	any	Another variable/value that should be compared.	Only if the input parameter is omitted.

Returns

TRUE or FALSE (see below).

Description

This operator compares two parameters. It returns TRUE if the first parameter is greater than or equal to the second parameter; otherwise FALSE will be returned. If the input parameter is provided, the operator will compare it with the first parameter; otherwise it is the first and the second parameter that will be compared. The following table shows how the different types are treated.

Type	Value
Number	The value of the number is used.
String	The number of characters is used.
Boolean	FALSE means 0 and TRUE means 1.
Array	The number of elements is used.
Object	The number of object attributes is used.
Other	Always 0.

Examples

Example 1

```
{256|ge( 128 )}
```

or

```
{ge( 256, 128 )}
```

Returns TRUE.

Example 2

```
{128|ge( 256 )}
```

or

```
{ge( 128, 256 )}
```

Returns FALSE.

Example 3

```
{256|ge( 256 )}
```

or

```
{ge( 256, 256 )}
```

Returns TRUE.

Example 4

```
{128|ge( 256, 64 )}
```

Returns FALSE.

gt

Summary

Returns TRUE if a parameter is greater than another parameter.

Usage

```
input|gt( value1 [, value2 ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared.	Yes.
value2	any	Another variable/value that should be compared.	Only if the input parameter is omitted.

Returns

TRUE or FALSE (see below).

Description

This operator compares two parameters. It returns TRUE if the first parameter is greater than the second parameter; otherwise FALSE will be returned. If the input parameter is provided, the operator will compare it with the first parameter; otherwise it is the first and the second parameter that will be compared. The following table shows how the different types are treated.

Type	Value
Number	The value of the number is used.
String	The number of characters is used.
Boolean	FALSE means 0 and TRUE means 1.
Array	The number of elements is used.
Object	The number of object attributes is used.
Other	Always 0.

Examples

Example 1

```
{256|gt( 128 )}
```

or

```
{gt( 256, 128 )}
```

Returns TRUE.

Example 2

```
{128|gt( 256 )}
```

or

```
{gt( 128, 256 )}
```

Returns FALSE.

Example 3

```
{256|gt( 256 )}
```

or

```
{gt( 256, 256 )}
```

Returns FALSE.

Example 4

```
{128|gt( 64, 256 )}
```

Returns TRUE.

le

Summary

Returns TRUE if a parameter is less than or equal to another parameter.

Usage

```
input|le( value1 [, value2 ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared.	Yes.
value2	any	Another variable/value that should be compared.	Only if the input parameter is omitted.

Returns

TRUE or FALSE (see below).

Description

This operator compares two parameters. It returns TRUE if the first parameter is less than or equal to the second parameter; otherwise FALSE will be returned. If the input parameter is provided, the operator will compare it with the first parameter; otherwise it is the first and the second parameter that will be compared. The following table shows how the different types are treated.

Type	Value
Number	The value of the number is used.
String	The number of characters is used.
Boolean	FALSE means 0 and TRUE means 1.
Array	The number of elements is used.
Object	The number of object attributes is used.
Other	Always 0.

Examples

Example 1

```
{256|le( 128 )}
```

or

```
{le( 256, 128 )}
```

Returns FALSE.

Example 2

```
{128|le( 256 )}
```

or

```
{le( 128, 256 )}
```

Returns TRUE.

Example 3

```
{256|le( 256 )}
```

or

```
{le( 256, 256 )}
```

Returns TRUE.

Example 4

```
{128|le( 256, 64 )}
```

Returns TRUE.

lt

Summary

Returns TRUE if a parameter is less than another parameter.

Usage

```
input|lt( value1 [, value2 ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared.	Yes.
value2	any	Another variable/value that should be compared.	Only if the input parameter is omitted.

Returns

TRUE or FALSE (see below).

Description

This operator compares two parameters. It returns TRUE if the first parameter is less than the second parameter; otherwise FALSE will be returned. If the input parameter is provided, the operator will compare it with the first parameter; otherwise it is the first and the second parameter that will be compared. The following table shows how the different types are treated.

Type	Value
Number	The value of the number is used.
String	The number of characters is used.
Boolean	FALSE means 0 and TRUE means 1.
Array	The number of elements is used.
Object	The number of object attributes is used.
Other	Always 0.

Examples

Example 1

```
{256|lt( 128 )}
```

or

```
{lt( 256, 128 )}
```

Returns FALSE.

Example 2

```
{128|lt( 256 )}
```

or

```
{lt( 128, 256 )}
```

Returns TRUE.

Example 3

```
{256|lt( 256 )}
```

or

```
{lt( 256, 256 )}
```

Returns FALSE.

Example 4

```
{128|lt( 256, 64 )}
```

Returns TRUE.

ne

Summary

Returns TRUE if one or more of the parameters do not match.

Usage

```
input|ne( value1 [, value2 [, ...] ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value that should be compared..	Yes.
value2	any	Another variable/value that should be compared.	Only if the input parameter is omitted.

Returns

TRUE or FALSE (see below).

Description

This operator compares all the provided parameters. If the parameters are not equal, the operator will return TRUE, otherwise FALSE will be returned. If more than one parameter is provided, the operator will ignore the input parameter.

Note that "ne" compares the values in the same way as the '!=> operator in PHP programming language (refer to the [PHP reference documentation](#) for more information). It is recommended to use the €compare (page 923)€ template operator for string comparision.

Examples

Example 1

```
{128|ne( 128 )}
```

or

```
{ne( 128, 128 )}
```

Returns FALSE.

Example 2

```
{128|ne( 256 )}
```

or

```
{ne( 128, 256 )}
```

Returns TRUE.

Example 3

```
{256|ne( 256, 128 )}
```

Returns TRUE.

Example 4

```
{ne( 128, 128, 256 )}
```

Returns TRUE.

Example 5

```
{ne( 128, 128, 128 )}
```

Returns FALSE.

not**Summary**

Returns the opposite of the input or the first parameter (TRUE/FALSE).

Usage

```
input|not( test )
```

Parameters

Name	Type	Description	Required
test	any	The variable/value that should be tested.	No.

Returns

TRUE or FALSE

Description

This operator returns TRUE if the input value is FALSE and vice versa. It is also possible to use the optional "test" parameter for evaluation. If both the input parameter and the "test" parameter are used, it is the input parameter that will be evaluated.

Examples**Example 1**

```
{if false()|not()
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The truth is out there.".

Example 2

```
{if true()|not()}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The day the earth stood still.".

Example 3

```
{if not( false() )}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The truth is out there.".

Example 4

```
{if not( true() )}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The day the earth stood still.".

null**Summary**

Returns TRUE if the input value is NULL (not the same as 0).

Usage

```
input|null()
```

Returns

TRUE or FALSE.

Description

This operator returns TRUE if the input value is NULL, otherwise FALSE will be returned. Please note that NULL is not the same as a numeric zero (0).

Examples**Example 1**

```
{if 0|null()}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The day the earth stood still".

or

Summary

Evaluates all parameters until one is found to be TRUE, returns that value.

Usage

```
or( value1 [, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
value1	any	A variable/value to be evaluated.	Yes.
value2	any	Another variable/value to be evaluated.	No.

Returns

One of the parameters or FALSE (see below).

Description

This operator evaluates all parameters until one of them is found to be TRUE. The operator will then return that parameter and thus stop evaluating the rest of the parameters. If none of the parameters are found to be TRUE, the operator will return FALSE. The following table shows how the different types are evaluated by this operator.

Type	Evaluation
Number	TRUE if the value is non-zero, FALSE otherwise.
String	TRUE if the string consists of at least one character, FALSE otherwise.
Boolean	TRUE if the boolean is a TRUE value, FALSE is otherwise.
Array	TRUE if the array has one or more elements, FALSE otherwise.
Object	TRUE if the object provides the "attributes" and the "attribute" methods, FALSE otherwise.
NULL	Always FALSE.
Other	Other types will be evaluated in the same way as PHP would do.

Examples

Example 1

```
{if or( false(), true(), false() )}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The truth is out there.".

Example 2

```
{def $a=array()
    $b=array( 1, 2, 3 )
    $c=array( 4, 5, 6 )}

{or( $a, $b, $c )}
```

The code above will return the following array: (1, 2, 3).

true

Summary

Creates and returns a boolean TRUE.

Usage

```
true()
```

Returns

TRUE.

Description

This operator creates and returns a boolean TRUE. It can be used to define a boolean variable and in logical comparisons.

Examples

Example 1

```
{def $my_boolean=true()}

{if $my_boolean}
    The truth is out there.
{else}
    The day the earth stood still.
{/if}
```

The following output will be produced: "The truth is out there.".

5.7.6 Mathematics

abs (page 866)

Returns a positive value of either the input or the first parameter.

ceil (page 868)

Returns the next highest integer value of input or parameter.

dec (page 870)

Returns input or parameter decremented by one.

div (page 872)

Divides input or first parameter by the remaining parameters.

floor (page 874)

Returns the next lowest integer value of input or parameter.

inc (page 876)

Increments either the input or the first parameter with one.

max (page 878)

Returns the largest value of all parameters.

min (page 879)

Returns the smallest value of all parameters.

mod (page 880)

Returns the modulo of two parameters.

mul (page 882)

Multiplies all parameters and returns the result.

round (page 884)

Returns a rounded version of the input or a parameter value.

sub (page 886)

Subtracts all remaining parameters from the first parameter.

sum (page 888)

Returns the sum of all parameters.

abs**Summary**

Returns a positive value of either the input or the first parameter.

Usage

```
input|abs( value )
```

Parameters

Name	Type	Description	Required
value	number	Value to calculate absolute of.	Only if the input parameter is omitted.

Returns

Absolute value of input or parameter.

Description

Returns a positive value of either the input or the "value" parameter. If both are provided, it is the "value" parameter that will be used.

Examples**Example 1**

```
{-16|abs}
```

or

```
{abs( -16 )}
```

The following output will be produced: "16".

Example 2

```
{abs( 256 )}
```

The following output will be produced: "256".

Example 3

```
{-64|abs( -128 )}
```

The following output will be produced: "128".

ceil

Summary

Returns the next highest integer value of input or parameter.

Usage

```
input|ceil( value )
```

Parameters

Name	Type	Description	Required
value	number	Value to be rounded up.	Only if the input parameter is omitted.

Returns

Integer (rounded up version of input/parameter).

Description

This operator returns the next highest integer value by rounding up either the input or the "value" parameter. If both are provided, it is the "value" parameter that will be used.

Examples

Example 1

```
{1.5|ceil}
```

or

```
{ceil( 1.5 )}
```

The following output will be produced: "2".

Example 2

```
{5.5|ceil( 8.2 )}
```

The following output will be produced: "9".

dec

Summary

Returns input or parameter decremented by one.

Usage

```
input|dec( value )
```

Parameters

Name	Type	Description	Required
value	number	The value that should be decremented.	Only if the input parameter is omitted.

Returns

input/parameter decremented with one.

Description

This operator decrements either the input or the "value" parameter with one and returns the result as an integer. If both are provided, it is the "value" parameter that will be used. Please note that this operator can not be used directly to decrement the value of a variable (please refer to the last example).

Examples

Example 1

```
{256|dec}
```

or

```
{dec( 256 )}
```

The following output will be produced: 255.

Example 2

```
{200|dec( 250 )}
```

The following output will be produced: 249.

Example 3

```
{def $i=256}  
{set $i=dec( $i )}  
{$i}
```

This example demonstrates how the value of a variable can be decremented using the "set" and the "dec" operators. The following output will be produced: "255".

div

Summary

Divides input or first parameter by the remaining parameters.

Usage

```
input|div( value1 [, value2 [, ...] ] )
```

Parameters

Name	Type	Description	Required
value1	number	Dividend or divisor.	Yes.
value2	number	Dividend.	Only if the input parameter is omitted.

Returns

The result of the division.

Description

This operator divides a the first parameter (can be the input parameter) by the rest of the parameters.

Examples

Example 1

```
{10|div( 5 )}
```

or

```
{div( 10, 5 )}
```

The following output will be produced: "2".

Example 2

```
{12|div( 3, 2 )}
```

or

```
{div( 12, 3, 2 )}
```

The following output will be produced: "2".

floor

Summary

Returns the next lowest integer value of input or parameter.

Usage

```
input|floor( value )
```

Parameters

Name	Type	Description	Required
value	number	Value to be rounded down.	Only if the input parameter is omitted.

Returns

Integer (rounded down version of input/parameter).

Description

This operator returns the next lowest integer by rounding down either the input or the "value" parameter. If both are provided, it is the "value" parameter that will be used.

Examples

Example 1

```
{256.7|floor}
```

or

```
{floor( 256.7 )}
```

The following output will be produced: "256".

Example 2

```
{999.2|floor( 256.7 )}
```

The following output will be produced: "256".

inc

Summary

Increments either the input or the first parameter with one.

Usage

```
input|inc( value )
```

Parameters

Name	Type	Description	Required
value	number	The value that should be incremented.	Only if the input parameter is omitted.

Returns

Number (input/parameter incremented with one).

Description

This operator increments either the input or the "value" parameter with one and returns the result as an integer. If both are provided, it is the "value" parameter that will be used. Please note that this operator can not be used directly to increment the value of a variable (please refer to the last example).

Examples

Example 1

```
{255|inc}
```

or

```
{inc( 255 )}
```

The following output will be produced: "256".

Example 2

```
{def $i=255}  
{set $i=inc( $i )}  
{$i}
```

The following output will be produced: "256".

max**Summary**

Returns the largest value of all parameters.

Usage

```
max( value1, value2 [,...] )
```

Parameters

Name	Type	Description	Required
value1	any	A value that should be evaluated.	Yes.
value2	any	Another value that should be evaluated.	Yes.

Returns

The largest value of all parameters.

Description

This operator returns the largest value of all parameters. The input parameter is ignored.

Examples**Example 1**

```
{max( 2, 3, 1 )}
```

The following output will be produced: "3".

Example 2

```
{max( array( 1, 2 ), array( 1, 2, 3 ) )}
```

The following array will be returned: (1, 2, 3).

min

Summary

Returns the smallest value of all parameters.

Usage

```
min( value1, value2 [,...] )
```

Parameters

Name	Type	Description	Required
value1	any	A value that should be evaluated.	Yes.
value2	any	Another value that should be evaluated.	Yes.

Returns

The smallest value of all parameters.

Description

This operator returns the largest value of all parameters. The input parameter is ignored.

Examples

Example 1

```
{min( 10, 20, 6, 40, 50 )}
```

The following output will be produced: "6".

Example 2

```
{min( array( 1, 2 ), array( 1, 2, 3 ) )}
```

The following array will be returned: (1, 2).

mod

Summary

Returns the modulo of two parameters.

Usage

```
input|mod( value1 [,value2] )
```

Parameters

Name	Type	Description	Required
value1	number	Divisor or dividend.	Yes.
value2	number	Divisor or dividend.	Only if the input parameter is omitted.

Returns

Integer (the modulo of the supplied parameters).

Description

This operator returns the modulo (rest after division) of the first parameter divided by the second. The operator can also take an input parameter. If the input parameter is used, then it will be divided by the first parameter (and thus the second parameter will be ignored).

Examples

Example 1

```
{5|mod( 3 )}
```

or

```
{mod( 5, 3 )}
```

The following output will be produced: "2".

Example 2

```
{6|mod( 3, 7 )}
```

The following output will be produced: "0".

mul

Summary

Multiplies all parameters and returns the result.

Usage

```
input|mul( value1 [, value2] [, ...] )
```

Parameters

Name	Type	Description	Required
value1	number	Multiplicand.	Yes.
value2	number	Multiplicand.	Only if the input parameter is omitted.

Returns

The result of the multiplication (integer or float).

Description

This operator multiplies all parameters and returns the result. If an input parameter is provided, it will be included in the multiplication.

Examples

Example 1

```
{2|mul( 3 )}
```

or

```
{mul( 2, 3 )}
```

The following output will be produced: "6".

Example 2

```
{2|mul( 3, 4 )}
```

The following output will be produced: "24".

round

Summary

Returns a rounded version of the input or a parameter value.

Usage

```
input|round( value )
```

Parameters

Name	Type	Description	Required
value	number	The number that should be rounded off.	Only if the input parameter is omitted.

Returns

A rounded off version of the provided value (an integer or float).

Description

This operator rounds off the value that was specified using either the input or the "value" parameter. If both are provided, it is the "value" parameter that will be used. The operator returns the rounded off value.

Examples

Example 1

```
{15.7|round}
```

or

```
{round( 15.7 )}
```

The following output will be produced: "16".

Example 2

```
{8.4|round( 9.7 )}
```

The following output will be produced: "10".

sub

Summary

Subtracts all remaining parameters from the first parameter.

Usage

```
input|sub( value [, ...] )
```

Parameters

Name	Type	Description	Required
value	number	A number that should be included in the subtraction.	Yes.

Returns

Number (result of subtraction).

Description

This operator subtracts all remaining parameters from the first parameter and returns the result. If an input parameter is provided, all other parameters will be subtracted from it.

Examples

Example 1

```
{10|sub( 2 )}
```

or

```
{sub( 10, 2 )}
```

The following output will be produced: "8".

Example 2

```
{sub( 10, 2, 3 )}
```

The following output will be produced: "5".

Example 3

```
{10|sub( 10, 2, 3 )}
```

The following output will be produced: "-5".

sum**Summary**

Returns the sum of all parameters.

Usage

```
input|sum( value [,...] )
```

Parameters

Name	Type	Description	Required
value	number	A value that should be added to the result.	Only if the input parameter is omitted.

Returns

Number (sum of all parameters + input).

Description

This operator adds up all the parameters (including the input parameter) and returns the result.

Examples**Example 1**

```
{1|sum( 2 )}
```

or

```
{sum( 1, 2 )}
```

The following output will be produced: "3".

Example 2

```
{1|sum( 2, -3, 4)}
```

The following output will be produced: "4".

5.7.7 Miscellaneous

action_icon (page 891)

Not documented yet.

attribute (page 892)

Makes it possible to inspect the contents of arrays, hashes and objects.

classgroup_icon (page 894)

Outputs an image tag referencing a class group icon.

class_icon (page 895)

Outputs an image tag referencing a class icon.

content_structure_tree (page 896)

Not documented yet.

ezpackage (page 897)

Not documented yet.

flag_icon (page 898)

Outputs an image tag referencing a flag icon.

gettime (page 899)

Converts a UNIX timestamp to a human friendly structure.

icon_info (page 901)

Not documented yet.

makedate (page 902)

Generates the UNIX timestamp of a given date.

maketime (page 903)

Generates the UNIX timestamp of a given date/time.

mimetype_icon (page 904)

Outputs an image tag referencing a MIME type icon.

month_overview (page 905)

Generates a structure that can be used to build a calendar.

pdf (page 907)

Provides access to the PDF functions.

roman (page 908)

Generates a roman representation of a number.

topmenu (page 909)

Not documented yet.

treemenu (page 910)

Fetches a subtree of nodes for the purpose of menu generation.

action_icon**Summary**

Not documented yet.

attribute**Summary**

Makes it possible to inspect the contents of arrays, hashes and objects.

Usage

```
input|attribute( [show_values [, level [, table ] ] ] )
```

Parameters

Name	Type	Description	Required
show_values	string	Extract values in addition to keys, names, etc.	No.
level	integer	The number of levels that should be processed (default is 2).	No.
table	boolean	Return result as HTML table (default) or not.	No.

Returns

A string revealing information about the target.

Description

This operator extracts all available keys, attribute names and/or methods that belong to the input parameter (must be either an object, an array or a hash). By default, only the array keys and object attribute names will be revealed. By passing "show" as the first parameter, the operator will also return the values. The second parameter can be used to control the number of levels/ children that should be expanded and included in the result (the default setting is 2). A large level value may cause the system to be trapped in a recursive/infinite loop. The returned result is an HTML table containing the retrieved information. If "false()" is passed as the third parameter, the output will be a plain string instead of an HTML table. Please refer to the "Array and object inspection" (page 201)section of the "Templates" chapter for more information about the use of this operator.

Examples**Example 1**

```
{def $example=hash( 'Name',   'John Doe',
                    'Age',     24,
                    'Phone',   '555-3212' )}
```

```
{$example|attribute()}
```

The following output will be produced:

Attribute	Type
Name	string
Age	integer
Phone	string

Example 2

```
{def $example=hash( 'Name', 'Jane Doe',
                    'Age', 23,
                    'Phone', '555-3213' )}

{$example|attribute( 'show' )}
```

The following output will be produced:

Attribute	Type	Value
Name	string	'Jane Doe'
Age	integer	23
Phone	string	'555-3213'

classgroup_icon

Summary

Outputs an image tag referencing a class group icon.

Usage

```
input|classgroup_icon( [ size [, alt_text [, return_uri ] ] ] )
```

Parameters

Name	Type	Description	Required
size	string	The preferred icon size (small, medium, large, etc.).	No.
alt_text	string	The alternative image text.	No.
return_uri	boolean	The return format (image tag or just the address).	No.

Returns

A string containing an image tag.

Description

This operator generates an image tag that references a class group icon. The name of the icon must be provided using the input parameter. The "size", "alt_text" and "return_url" parameters are optional (see the description above). The operator uses the settings provided by the "icon.ini" configuration file (or an override).

Examples

Example 1

```
{'content'|classgroup_icon( 'small', 'Alternative image text' )}
```

The following output will be produced:

```

```

class_icon

Summary

Outputs an image tag referencing a class icon.

Usage

```
input|class_icon( [ size, [, alt_text ] ] )
```

Returns

A string containing an image tag.

Description

This operator generates an image tag that references a class icon. The name of the icon must be provided using the input parameter. The "size" and "alt_text" parameters are optional (see the description above). The operator uses the settings provided by the "icon.ini" configuration file (or an override).

Examples

Example 1

```
{'folder'|class_icon( 'small', 'Alternative image text' )}
```

The following output will be produced:

content_structure_tree**Summary**

Not documented yet.

ezpackage**Summary**

Not documented yet.

flag_icon

Summary

Outputs an image tag referencing a flag icon.

Usage

```
input|flag_icon()
```

Returns

A string containing an image tag.

Description

This operator generates an image tag that references a flag icon. The country code must be provided using the input parameter. This operator is frequently used by the administration interface.

Examples

Example 1

```
{'eng-GB'|class_icon()}
```

The following output will be produced:

```

```

gettime

Summary

Converts a UNIX timestamp to a human friendly structure.

Usage

```
gettime( timestamp )
```

Parameters

Name	Type	Description	Required
timestamp	integer	A UNIX timestamp.	Yes.

Returns

An associative array (see below).

Description

This operator takes a UNIX timestamp as a parameter and returns an associative array that contains a human friendly representation of the provided timestamp. The following table shows the keys of the returned hash.

Key	Type	Description
seconds	integer	Number of seconds.
minutes	integer	Number of minutes.
hours	integer	Number of hours.
day	integer	The day of the month (1-31).
month	integer	The month number (1 is January).
year	integer	A four digit representation of the year (for example "1978").
weekday	integer	The day of the week (1-7).
weeknumber	string	The week of the year (1-52).
yearday	integer	The day of the year (1-365).
epoch	integer	The UNIX timestamp that was provided.

Examples

Example 1

```
{gettime( currentdate() )|attribute(show)}
```

The "currentdate" (page 806) operator is used to generate the UNIX timestamp for the current date/time. This value is then fed to the "gettime" operator, which converts it to a human friendly structure. The array is viewed using the "attribute" (page 892) operator. The following output will be produced:

Attribute	Type	Value
seconds	integer	11
minutes	integer	5
hours	integer	15
day	integer	13
month	integer	4
year	integer	2005
weekday	integer	4
weeknumber	string	15
yearday	integer	102
epoch	integer	1113397511

icon_info**Summary**

Not documented yet.

makedate

Summary

Generates the UNIX timestamp of a given date.

Usage

```
makedate( [month [, day [, year [, dst ] ] ] ] )
```

Parameters

Name	Type	Description	Required
month	integer	The month of the year.	No.
day	integer	The day of the month.	No.
year	integer	The year.	No.
dst	integer	Daylight savings (on/off).	No.

Returns

A UNIX timestamp (an integer).

Description

This operator returns the UNIX timestamp corresponding to the provided parameters. The parameters may be left out in order from right to left. Parameters that are omitted will be set to the current value according to the local date and time. The "dst" parameter can be set to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown whether the time is within daylight savings time or not (the system will try to figure it out). If no parameters are given, the operator will return the timestamp for the current day.

Examples

Example 1

```
{makedate( 1, 2, 2005 )}
```

The following output will be produced: "1104620400" - which is the UNIX timestamp for "00:00:00, 2nd of January, 2005".

maketime

Summary

Generates the UNIX timestamp of a given date/time.

Usage

```
maketime( [hour [, minute [, second [, month [, day [, year [, dst ] ] ] ] ] ] ] )
```

Parameters

Name	Type	Description	Required
hour	integer	Hour of the day.	No.
minute	integer	Minute of the hour.	No.
second	integer	Second of the minute.	No.
month	integer	Month of the year.	No.
day	integer	Day of the month.	No.
year	integer	The year.	No.
dst	integer	Daylight savings (on/off).	No.

Returns

A UNIX timestamp (an integer).

Description

This operator returns the UNIX timestamp corresponding to the provided parameters. The parameters may be left out in order from right to left. Parameters that are omitted will be set to the current value according to the local date and time. The "dst" parameter can be set to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown whether the time is within daylight savings time or not (the system will try to figure it out). If no parameters are given, the operator will return the current timestamp.

Examples

Example 1

```
{maketime( 1, 2, 3, 4, 5, 2004 )}
```

The following output will be produced: "1081119723" - which is the UNIX timestamp for "01:02:03, 5th of April, 2004".

mimetype_icon

Summary

Outputs an image tag referencing a MIME type icon.

Usage

```
input|mimetype_icon( [ size, [ , alt_text [, return_type ] ] ] )
```

Parameters

Name	Type	Description	Required
size	integer	The size of the icon (small, normal, large, etc.).	No.
alt_text	string	The alternative image text.	No.
return_url	boolean	The return type (FALSE=image tag, TRUE=address only).	No.

Returns

A string containing an image tag.

Description

This operator generates an image tag that references a MIME type icon. The name of the icon must be provided using the input parameter. The "size" and "alt_text" parameters are optional (see the description above). The operator uses the settings provided by the "icon.ini" configuration file (or an override).

Examples

Example 1

```
{'application/pdf'|mimetype_icon( 'small', 'Alternative image text' )}
```

This will output the MIME type icon that is associated with the ".pdf" file extension (defined in "icon.ini" or an override).

month_overview

Summary

Generates a structure that can be used to build a calendar.

Usage

```
input|month_overview( field,
                      today_timestamp,
                      hash( 'current',           current_timestamp,
                            'day_class',        day_class
                            'current_class',   current_class,
                            'link',            link,
                            'month_link',      month_link,
                            'year_link',       year_link,
                            'day_link',        day_link,
                            'next',            hash( 'link', next_link ),
                            'previous',        hash( 'link', previous_link ) ) )
```

Parameters

Name	Type	Description	Required
field	string	The way the objects should be grouped.	Yes.
today_timestamp	integer	The UNIX timestamp for the day being searched.	Yes.
current_timestamp	integer	The UNIX timestamp of the current date/time.	No.
day_class	string	The CSS class that should be used for a normal day.	No.
current_class	string	The CSS class that should be used for the current day.	No.
link	string	Link to days containing objects ("content/view/full/node_id").	No.
month_link	string	Append "/month/MON" to links or not.	No.
year_link	string	Append "/year/YEAR" to links or not.	No.
day_link	string	Append "/day/DAY" to links or not.	No.
next_link	string	Link to the next month.	No.
previous_link	string	Link to the previous month.	No.

Returns

A complex structure (see below).

Description

This operator takes an array of content objects as input and analyzes and distributes them in a given month. It returns a complex structure (see below) that can be used by the "monthview.tpl" template, which will highlight days when at least one object has been published (see "design/standard/templates/navigator/monthview.tpl"). The following structure will be returned:

```
.  
|-- year  
|-- month  
|-- weekdays []  
|  |-- day  
|  |-- class  
|  
|-- weeks [] []  
|  |--day  
|  |--link  
|  |--class  
|  |--highlight  
|  
|-- next  
|  |--month  
|  |--link  
|  
|-- previous  
|  |--month  
|  |--link
```

pdf

Summary

Provides access to the PDF functions.

Usage

```
{pdf( ... )}
```

Returns

The returned result depends on the actual PDF function.

Description

This operator provides access to the PDF functions that may be used to generate PDF data in a template. The actual PDF functions are documented in the "Template PDF functions" (page [1089](#)) section of the "Reference" chapter.

roman

Summary

Generates a roman representation of a number.

Usage

```
input|roman( value )
```

Parameters

Name	Type	Description	Required
value	integer	A number that should be converted.	Only if the input parameter is omitted.

Returns

A string containing a roman number.

Description

This operator will convert either the input or the "value" parameter to a roman number. If both are provided, it is the "value" parameter that will be used.

Examples

Example 1

```
{8|roman()}
```

or

```
{roman( 8 )}
```

The following output will be produced: "VIII".

topmenu**Summary**

Not documented yet.

treemenu**Summary**

Fetches a subtree of nodes for the purpose of menu generation.

Usage

```
treemenu( path,
          node_id
            [, class_filter      ]
            [, depth_skip       ]
            [, max_level        ]
            [, is_selected_method ] )
```

Parameters

Name	Type	Description	Required
path	array	An array of the path (\$module_result.path).	Yes.
node_id	integer	The node ID number (the root of the subtree).	Yes.
class_filter	array	An array of classes that should be filtered.	No.
depth_skip	integer	Number of levels that should be skipped.	No.
max_level	integer	The max depth that should be explored. (2 by default)	No.
is_selected_method	string	Sets whether "is_selected=TRUE" should be assigned to the parents of the current node as well.	No.

Returns

A complex structure that can be used to build a menu (see below).

Description

This operator fetches a subtree of nodes and returns a complex structure (an array of hashes) that can be used to generate a menu.

If the optional "class_filter" parameter is omitted, all nodes will be fetched without filtering. If an empty array is passed, then only folder nodes (class ID = 1) will be fetched.

The default value of the "is_selected_method" parameter is "tree". This value determines that the current node and its parent nodes will be considered as selected (is_selected=TRUE). If this parameter is set to "node", only the current node will be considered as selected.

The following table shows the hash-structure for each element in the array that will be returned.

Key	Type	Description
id	integer	The ID of the node.
level	integer	The depth of the node.
url_alias	string	The URL alias of the node.
url	string	The system URL of the node.
text	string	The name of the node.
is_selected	boolean	TRUE if the node is currently being viewed/selected, FALSE otherwise.

Examples

Example 1 (explanatory)

Let's use a small site with the following content structure (see the screenshot) for the purpose of demonstration

(see figure 5.78)

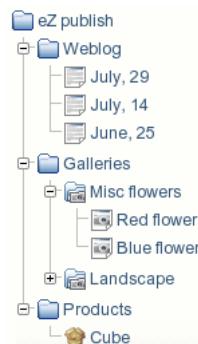


Figure 5.78: The content tree

and insert the following code into "pagelayout.tpl":

```

{def $mainMenu=treemenu( $module_result.path, $module_result.node_id ) }
{foreach $mainMenu as $menu}
{$menu.level} - {$menu.text}<br />
{/foreach}
{undef $mainMenu}
  
```

If the "Weblog" node is being viewed, then the following output will be produced:

```
0 - Weblog  
1 - July, 29  
1 - July, 14  
1 - June, 25  
0 - Galleries  
0 - Products
```

If the "Blue flower" node is being viewed, then the following output will be produced:

```
0 - Weblog  
0 - Galleries  
1 - Misc flowers  
1 - Landscape  
0 - Products
```

Since the "max_level" parameter is omitted, only two levels are explored. To set the "max_level" parameter to 3, change the first line of the previous code fragment in the following way:

```
{def $mainMenu=treemenu( $module_result.path, $module_result.node_id, , , 3 ) }
```

This will produce the following output for the "Blue flower" node:

```
0 - Weblog  
0 - Galleries  
1 - Misc flowers  
2 - Red flower  
2 - Blue flower  
1 - Landscape  
0 - Products
```

To skip the first level, set the "depth_skip" parameter to 1 by changing the first line of the code fragment as shown below:

```
{def $mainMenu=treemenu( $module_result.path, $module_result.node_id, , 1, 3 ) }
```

If the "Blue flower" node is being viewed, then the following output will be produced:

```
0 - Misc flowers  
1 - Red flower  
1 - Blue flower  
0 - Landscape
```

Now, let's use another code fragment in order to see which items are selected:

```
{def $mainMenu=treemenu( $module_result.path, $module_result.node_id, , , 3 )}  
{foreach $mainMenu as $menu}  
    {if $menu.is_selected}  
        {$menu.level} - {$menu.text} (selected) <br />  
    {else}  
        {$menu.level} - {$menu.text} <br />  
    {/if}  
{/foreach}  
{undef $mainMenu}
```

Since the "is_selected_method" parameter is omitted, the "tree" mode will be used and the following output will be produced for the "Blue flower" node:

```
0 - Weblog  
0 - Galleries (selected)  
1 - Misc flowers (selected)  
2 - Red flower  
2 - Blue flower (selected)  
1 - Landscape  
0 - Products
```

To set the "is_selected_method" parameter to "node", replace the first line of the last code fragment by the following line:

```
{def $mainMenu=treemenu( $module_result.path, $module_result.node_id, , , 3,  
'node' )}
```

If the "Blue flower" node is being viewed, then the following output will be produced:

```
0 - Weblog  
0 - Galleries  
1 - Misc flowers  
2 - Red flower  
2 - Blue flower (selected)  
1 - Landscape  
0 - Products
```

Example 2

These examples are from pagelayout.tpl.

Make a menu of folder and info_page classes. Skip the first level and maximum go to depth 6.

```
<ul>
{def $mainMenu=treemenu( $module_result.path,
                         $module_result.node_id,
                         array('folder','info_page'), 1, 6 )}

{foreach $mainMenu as $menu}
    <li class="level_{$menu.level}">

        {if $menu.is_selected}
            <div class="selected">
                <a href={$menu.url_alias|ezurl}>{$menu.text}</a>
            </div>
        {else}
            <a href={$menu.url_alias|ezurl}>{$menu.text}</a>
       {/if}

    </li>
{/foreach}
</ul>
```

Make a menu which shows the sub menu items when clicked on the parent menu item. Notice that only the objectclass ids: 1, 9, and 17 are visible.

```
{def $docs=treemenu( $module_result.path, $module_result.node_id,
                      array(1, 9, 17), 0, 4)}
{def $depth=1 $last=0}

{foreach $docs as $menu}
    {if and($last != 0, $last.level > $menu.level)}
        </ul>
        </li>
   {/if}

    <li>

        {if and($last != 0, $last.level < $menu.level)}
            <ul>
            <li>
       {/if}

        <a {$menu.is_selected|choose('','class="selected')} href={$menu.url_alias|ezurl}>{$menu.text|shorten(25)}</a>
        </li>

        {set last=$menu}
   {/foreach}
```

```
{while $depth |gt(1)}  
  </li>  
  </ul>  
  {set depth=$depth|sub(1)}  
{/while}
```

5.7.8 Strings

```
\begin{description}           \item[append]\refer{labe1ff63e0a715b9556358feb062efa6b66d}\\\ Returns the input string with a custom sequence appended to it.\item[autolink]\refer{labele6e9efb58d74fa146620748b0ac4f089}\\\ Returns the input string with the addresses replaced by link tags.\item[begins\_with]\refer{labelda574a5c074ab5f450257fb022aa628e}\\\ Checks if a string starts with a specific character/sequence.\item[break]\refer{labelea9f20146299d6a77d7bec65149c644e}\\\ Returns the input string with all newlines converted to HTML breaks.\item[chr]\refer{labele7d37963373bad3659db02913efcf3bb}\\\ Generates a string based on the input array of ASCII/UNICODE values.\item[compare]\refer{label1526b6230bdbe7fa3db0484c2963024}\\\ Compares the contents of two strings.\item[concat]\refer{labelb036f6fee0f64d95787d7d240904e5ec}\\\ Merges several strings into one string.\item[contains]\refer{labeld7ba916a2d323da38c2fbece6b1ad254}\\\ Checks if a string contains a specific element.\item[count\_chars]\refer{labele08262e0efcc4867b3436207c2e949da}\\\ Returns the length of the input string.\item[count\_words]\refer{label60d8245e2361648a8e03edbc99803a51}\\\ Returns the number of words that make up the input string.\item[crc32]\refer{label1e19d8bc566dc8bef851c52d6a676520}\\\ Returns the CRC32 polynomial of the input string.\item[downcase]\refer{labela9ce08bf94b5a3eb8424090ee6fb3b8f}\\\ Returns a lowercased version of the input string.\item[ends\_with]\refer{label9e94bd95ca9d93ea5e44fa4d1146dbc5}\\\ Checks if a string ends with a specific character/sequence.\item[explode]\refer{labelbd329d9cef61c18dd13b374aaadd03a}\\\ Splits the input string and returns it as an array of strings.\item[extract]\refer{label882b1a718fe78d78671ea1af4af030b0}\\\ Returns a portion of the input string.\item[extract\_left]\refer{labela5fdbfa799b416f9c5d8e643ed62fd63}\\\ Returns a portion of the start of the input string.\item[extract\_right]\refer{label429b1fcaa4f401dfb7f1dd2514bb}\\\ Returns a portion of the end of the input string.\item[indent]\refer{label7f7dd323926487eac8ed79061a742e67}\\\ Returns an indented version of the input string.\item[insert]\refer{label62af080e7f7bfc0be55bcb520e44a8e6}\\\ Returns the input string with additional text inserted at a specified position.\item[md5]\refer{label0cc133b83534e32be42372e5f849c730}\\\ Returns the MD5 hash of the input string.\item[nl2br]\refer{labelb143344ea25c81db2141b12f0ad8ba85}\\\ Returns the input string with all newlines converted to HTML breaks.\item[ord]\refer{label20776555526334033bff577906930aa8}\\\ Returns an array containing the ASCII/UNICODE values of the input string.\item[pad]\refer{label2be76638f3804b74529d8afa36eec027}\\\ Returns a lengthened version of the input string.\item[prepend]\refer{label64f2fb0180e120f843bf5cbefde6fb2}\\\ Returns the input string prepended with a custom sequence.\item[remove]\refer{labelff98c2820a6fef7761ea0543dc1dc226c}\\\ Returns a pruned version of the input string.\item[repeat]\refer{labelcb19098c71994b6d7180ddca9b69f7c9}\\\ Returns a repeated version of the input string.\item[reverse]\refer{labela65b19ab58d49f7d0d8d30507dfbe6d}\\\ Returns a reversed version of the input string.\item[rot13]\refer{label257f45cf216499bcb93d753dbc915}\\\ Returns a ROT13 transformation of the input string.\item[shorten]\refer{label4adeb28946205c4ab19e7adb9bf4d617}\\\ Returns a shortened ver-
```

s
S
information

sion of the input string. \item[simpletags]\refer{labelf36d8ad532132910c4c45869b5e9a163}\ \\ Returns a partially marked up version of the input string. \item[simplify]\refer{label1dc4ba030b96e65d66c7e044be68b152}\ \\ Returns a simplified version of the input string. \item[trim]\refer{labelf1f41f3b519b1ad3cdedb820318f0689}\ \\ Returns a stripped version of the input string. \item[upcase]\refer{labelaa49a6422936e6706335b2481dcad}\ \\ Returns a capitalized version of the input string. \item[upfirst]\refer{label30c5b91fa9902052abbc7f70cb}\ \\ Returns the input string with a capitalized initial letter. \item[upword]\refer{labelfd18620cb58f740bc736dc7484df82eb}\ \\ Returns the input string with capitalized initial letters. \item[wash]\refer{labela57f0869e67aa55eed1c2c95d29f2024}\ \\ Returns an HTML-safe version of the input string. \item[wordtoimage]\refer{labelcbb9f9c9a683669158c28f061d11f730}\ \\ Returns the input string with embedded image tags. \item[wrap]\refer{label0551ef7ee169e08f4d0709b33d9152e5}\ \\ Returns a wrapped version of the input string.

\end{description}

append

Summary

Returns the input string with a custom sequence appended to it.

Usage

```
input|append( value1 [, value2 [, ... ] ] )
```

Parameters

Name	Type	Description	Required
value1	mixed	Text to be appended.	Yes.
value2	mixed	More text that should be appended.	No.

Returns

A string consisting of the input string and the parameters.

Description

This operator appends the parameter value(s) at the end of the input string and returns the resulting string.

Examples

Example 1

```
{'The '|append( 'Last ', 'Crusade ')}
```

The following output will be produced: "The Last Crusade".

autolink

Summary

Returns the input string with the addresses replaced by link tags.

Usage

```
input|autolink()
```

Returns

A string with marked up links.

Description

This operator takes a string as an input. It will convert links and E-mail addresses to clickable entities using HTML markup. The resulting string (containing markup here and there) will be returned.

Examples

Example 1

```
{'Blah blah http://www.example.com blah hello@example.com blah.'|autolink()}
```

The following output will be produced:

```
Blah blah <a href="http://www.example.com">http://www.example.com</a> blah <a href="mailto:hello@example.com">hello@example.com</a> blah.
```

begins_with

Summary

Checks if a string starts with a specific character/sequence.

Usage

```
input|begins_with( sequence )
```

Parameters

Name	Type	Description	Required
sequence	string	The string that should be matched.	Yes.

Returns

TRUE if there is a match, FALSE otherwise.

Description

This operator checks if the input string starts with a specified sequence of characters. If yes, the operator returns TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{'My cat is green.'|begins_with( 'My cat' )}
```

Returns TRUE.

Example 2

```
{'My cat is green.'|begins_with( 'My dog' )}
```

Returns FALSE.

break**Summary**

Returns the input string with all newlines converted to HTML breaks.

Usage

```
input|break()
```

Returns

A string with HTML breaks.

Description

This operator takes a string as input. It does not replace newline characters/sequences, but it does insert HTML break tags (`
`) before all newlines in a string and returns a modified version of the input.

Examples**Example 1**

```
{'The lazy  
cat  
jumps over  
the quick rat.'|break()}
```

The following output will be produced:

```
The lazy<br />cat<br />jumps over<br />the quick rat.
```

chr

Summary

Generates a string based on the input array of ASCII/UNICODE values.

Usage

```
input|chr()
```

Returns

A string containing the requested characters.

Description

This operator creates and returns a string. The contents of the returned string is determined by the input parameter which must be an array of ASCII/UNICODE values (as integers).

Examples

Example 1

```
{array( 97, 98, 99 )|chr()}
```

The following output will be produced: "abc".

compare

Summary

Compares the contents of two strings.

Usage

```
input|compare( compare_with )
```

Returns

TRUE if the strings match, FALSE otherwise.

Description

This operator compares the contents of two strings and returns TRUE if they're identical, FALSE if they differ.

Examples

Example 1

```
{'Hello world'|compare( 'Hello world' )}
```

Returns TRUE.

Example 2

```
{'Hello world'|compare( 'Goodbye world' )}
```

Returns FALSE.

concat**Summary**

Merges several strings into one string.

Usage

```
concat( value1, value2 [,...] )
```

Parameters

Name	Type	Description	Required
value1	mixed	A string that should be added.	Yes.
value2	mixed	Another string that should be added.	Yes.

Returns

A string consisting of all the parameters.

Description

This operator merges several strings into one and returns the resulting string.

Examples**Example 1**

```
{concat( 'what', 'ever' )}
```

The following output will be produced: "whatever".

Example 2

```
{def $number=256}
{concat( 'The number is: ', $number, '!' )}
```

The following output will be produced: "The number is: 256!"

contains

Summary

Checks if a string contains a specific element.

Usage

```
input|contains( sequence )
```

Parameters

Name	Type	Description	Required
sequence	string	The string that should be matched.	Yes.

Returns

TRUE if there is a match, FALSE otherwise.

Description

This operator checks if the input string contains a specific sequence of characters. If a match is found, the operator will return TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{'Welcome to my homepage!'|contains('my')}
```

Returns TRUE.

Example 2

```
{'Welcome to my homepage!'|contains('your')}
```

Returns FALSE.

count_chars

Summary

Returns the length of the input string.

Usage

```
input|count_chars()
```

Returns

An integer revealing the string length.

Description

This operator counts and returns the number of characters (all of them, whitespaces included) that make up the input string.

Examples

Example 1

```
{'Testing 1 2 3'|count_chars()}
```

The following output will be returned: "13".

Example 2

```
{'Testing'|count_chars()}
```

The following output will be returned: "7".

count_words**Summary**

Returns the number of words that make up the input string.

Usage

```
input|count_words()
```

Returns

An integer revealing the number of words that make up a string.

Description

This operator counts and returns the number of words that are found within the input string.

Examples**Example 1**

```
{'Where do you want to publish today?'|count_words()}
```

The following output will be returned: "7".

crc32**Summary**

Returns the CRC32 polynomial of the input string.

Usage

```
input|crc32()
```

Returns

The CRC32 polynomial of the input string.

Description

This operator calculates and returns the CRC32 polynomial of the input string.

Examples**Example 1**

```
{'filename.txt'|crc32()}
```

The following output will be produced: "-460339180".

downcase**Summary**

Returns a lowercased version of the input string.

Usage

```
input|downcase()
```

Returns

A lowercase version of the input string.

Description

This operator returns a lowercased version of the input string.

Examples**Example 1**

```
{"My StriNG Is CoOl!"|downcase()}
```

The following output will be produced: "my string is cool!".

ends_with

Summary

Checks if a string ends with a specific character/sequence.

Usage

```
input|ends_with( sequence )
```

Parameters

Name	Type	Description	Required
sequence	string	The string that should be matched.	Yes.

Returns

TRUE if there is a match, FALSE otherwise.

Description

This operator checks if the input string ends with a specified sequence of characters. If yes, the operator returns TRUE, otherwise FALSE will be returned.

Examples

Example 1

```
{'Linux is great!'}|ends_with( 'great!' )
```

Returns TRUE.

Example 2

```
{'Linux is great!'}|begins_with( 'great' )
```

Returns FALSE.

explode

Summary

Splits the input string and returns it as an array of strings.

Usage

```
input|explode( separator )
```

Parameters

Name	Type	Description	Required
separator	string	Split sequence.	Yes.

Returns

An array of strings.

Description

This operator takes a string as input and returns an array of strings. Each element in the array will be a part of the input string extracted on the basis of the specified sequence of split characters.

Examples

Example 1

```
{'All-your-base-are-belong-to-us!'}|explode( '-' )}
```

The following array will be returned: ('All', 'your', 'base', 'are', 'belong', 'to', 'us!').

extract**Summary**

Returns a portion of the input string.

Usage

```
input|extract( offset [, length] )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	Yes.
length	integer	The number of characters that should be extracted.	No.

Returns

A string containing a portion of the input string.

Description

This operator will return a portion of the input string. The returned portion must be defined by the "offset" and "length" parameters. If the "length" parameter is omitted, the rest of the string (from offset) will be returned.

Examples**Example 1**

```
{'I love monday mornings!'}|extract( 7 )}
```

The following output will be produced: "monday mornings!".

Example 2

```
{'Big apples.'|extract( 4, 5 )}
```

The following output will be produced: "apple".

extract_left

Summary

Returns a portion of the start of the input string.

Usage

```
$input_string|extract_left( length )
```

Parameters

Name	Type	Description	Required
length	integer	The number of characters that should be extracted.	Yes.

Returns

A string containing a chunk of the input string.

Description

This operator extracts a portion from the start of the input string. The "length" parameter must be used to define the length (number of characters) of the portion.

Examples

Example 1

```
{'Gooooood morning Vietnam!'|extract_left( 8 )}
```

The following output will be produced: "Gooooood".

extract_right

Summary

Returns a portion of the end of the input string.

Usage

```
input|extract_right( length )
```

Parameters

Name	Type	Description	Required
length	integer	The number of characters that should be extracted.	Yes.

Returns

A string containing a chunk of the input string.

Description

This operator extracts a portion from the end of the input string. The "length" parameter must be used to define the length (number of characters) of the portion.

Examples

Example 1

```
{"Gooooood morning Vietnam!"|extract_right( 8 )}
```

The following output will be produced: "Vietnam!".

indent

Summary

Returns an indented version of the input string.

Usage

```
input|indent( count [, type [, filler ] ] )
```

Parameters

Name	Type	Description	Required
count	integer	Number of indentations.	Yes.
type	string	Type of indentation ("space", "tab" or "custom").	No.
filler	string	Custom indentation.	No.

Returns

An indented version of the input string.

Description

This operator indents the input string and returns it. The indentation type can be set to "space", "tab" or "custom". The default indentation is "space". If the indentation type is set to "custom", the "filler" parameter must be set to the desired indentation string.

Examples

Example 1

```
{'This is my text'|indent( 1 )}
```

The following string will be returned: " This is my text".

Example 2

```
{'This is my second line'|indent( 3, 'custom', '.' )}
```

The following string will be returned: "...This is my second line".

insert**Summary**

Returns the input string with additional text inserted at a specified position.

Usage

```
input|insert( offset, sequence )
```

Parameters

Name	Type	Description	Required
offset	integer	The position where the sequence should be inserted.	Yes.
sequence	string	The string that should be inserted.	Yes.

Returns

A string consisting of the input and the inserted sequence.

Description

This operator inserts a sequence of characters at a specified position of the input string. The resulting string will be returned.

Examples**Example 1**

```
{'My string is simple.'|insert( 3, 'static ' )}
```

The following output will be produced: "My static string is simple.".

md5

Summary

Returns the MD5 hash of the input string.

Usage

```
input|md5()
```

Returns

The MD5 hash of the input string.

Description

This operator calculates and returns the MD5 hash of the input string.

Examples

Example 1

```
{'Desktop computer'|md5()}
```

The following output will be produced: "8cae7108f44d1958b9febc65e44cbbc8".

nl2br

Summary

Returns the input string with all newlines converted to HTML breaks.

Usage

```
input|nl2br()
```

Returns

A string with HTML breaks.

Description

This operator takes a string as input. It replaces newline characters/sequences with HTML break tags and returns a modified version of the input.

Examples

Example 1

```
{'The lazy  
cat  
jumps over  
the quick rat.'|break()}
```

The following output will be produced:

```
The lazy<br />cat<br />jumps over<br />the quick rat.
```

ord**Summary**

Returns an array containing the ASCII/UNICODE values of the input string.

Usage

```
input|ord()
```

Returns

An array with ASCII/UNICODE values.

Description

This operator returns an array containing the ASCII/UNICODE values of the characters that make up the input string.

Examples**Example 1**

```
{'abcdef'|ord()}
```

The following array will be returned: (97, 98, 99, 100, 101, 102).

pad

Summary

Returns a lengthened version of the input string.

Usage

```
input|pad( length [, padding ] )
```

Parameters

Name	Type	Description	Required
length	integer	The desired length of the string.	Yes.
padding	character	Custom character to be used for padding. (Default is space)	No.

Returns

A padded version of the input string.

Description

This operator makes sure that the input string is at least "length" characters long by inserting extra characters at the end. It is possible to specify a custom character using the "padding" parameter (default is space). The operator returns a padded version of the input string.

Examples

Example 1

```
{'Too short!'}|pad( 15 )}
```

The following string will be produced: "Too short! ".

Example 2

```
{'Too short!'}|pad( 16, '-' )}
```

The following string will be produced: "Too short!---".

prepend

Summary

Returns the input string prepended with a custom sequence.

Usage

```
input|prepend( sequence )
```

Parameters

Name	Type	Description	Required
sequence	string	The string that should be prepended.	Yes.

Returns

A string consisting of the first parameter and the input string.

Description

This operator puts the "sequence" parameter at the start of the input string and returns the resulting string.

Examples

Example 1

```
{'Weaver'|prepend('Sigourney')}
```

The following string will be produced: "Sigourney Weaver".

remove**Summary**

Returns a pruned version of the input string.

Usage

```
input|remove( offset, length )
```

Parameters

Name	Type	Description	Required
offset	integer	The offset to start at.	Yes.
length	integer	The number of characters that should be removed.	Yes.

Returns

A pruned version of the input string.

Description

This remove operator removes characters from the input string and returns the pruned version. The "offset" and "length" parameters must be used to define the start and length of the portion that should be removed.

Examples**Example 1**

```
'My string is simple.'|remove( 3, 2 )
```

The following string will be produced: "My ring is simple.".

repeat**Summary**

Returns a repeated version of the input string.

Usage

```
input|repeat( count )
```

Parameters

Name	Type	Description	Required
count	integer	The number of repeats.	Yes.

Returns

A repeated version of the input string.

Description

This operator returns a repeated version of the input string. The "count" parameter must be used to define the desired number of repetitions.

Examples**Example 1**

```
{'*DJ Cat Show*',|repeat( 2 )}
```

The following string will be produced: "*DJ Cat Show* *DJ Cat Show*".

reverse**Summary**

Returns a reversed version of the input string.

Usage

```
input|reverse()
```

Returns

A reversed version of the input string.

Description

This operator returns a reversed version of an input string.

Examples**Example**

```
{"Hello World" |reverse}
```

The following output will be produced: "dlroW olleH".

rot13**Summary**

Returns a ROT13 transformation of the input string.

Usage

```
input|rot13()
```

Returns

A rotated version of the input string.

Description

Returns a ROT13 transformation of the input string.

Examples**Example 1**

```
{'Hello World'|rot13()}
```

The following output will be produced: "Uryyb Jbeyq".

Example 2

```
{'Uryyb Jbeyq'|rot13()}
```

The following output will be produced: "Hello world".

shorten

Summary

Returns a shortened version of the input string.

Usage

```
input|shorten( [ length [, sequence [, trim_type ] ] ] )
```

Parameters

Name	Type	Description	Required
length	integer	The desired length of the returned string.	No.
sequence	string	Custom trailing/end-sequence.	No.
trim_type	string	Controls the type of trimming: "right" (default) or "middle".	No.

Returns

A shortened version of the input string.

Description

This operator shortens the input string to "length" characters and adds a trailing sequence. Please note that the "length" parameter also includes the length of the trailing sequence. If the input string is shorter than "length", it will not be shortened. The default length is 80, the default trailing sequence is three dots: "...". The third parameter controls the type of trimming, it can be set to either "right" (default) or "middle".

Examples

Example 1

```
{'Led Zeppelin rocks!'|shorten( 15 )}
```

The following output will be produced: "Led Zeppelin...".

Example 2

```
{"eZ Systems"|shorten( 7, '...', 'middle' )}
```

The following output will be produced: "eZ...ms".

simpletags

Summary

Returns a partially marked up version of the input string.

Usage

```
input|simpletags( [ taglist ] )
```

Parameters

Name	Type	Description	Required
taglist	string	The name of the custom tag group that should be used.	No.

Returns

A partially marked up version of the input string.

Description

This operator returns a partially marked up version of the input string. It can be used to allow the usage/pass-through of a small subset of HTML/custom tags (other/disallowed tags will be removed). This operator is typically useful when it comes to allowing some kind of formatting in for example comments (instances of a class that does not support formatting through the XML block datatype). The optional "taglist" parameter can be used to select the list of allowed tags (the default is "TagList"). The tags/groups must be defined in a configuration override for "template.ini". The following table shows the tags that are allowed by default.

Custom tag	HTML replacement
literal	<pre>...</pre>
code	<pre class="code">...</pre>
strong	...
emphasize	<i>...</i>

Examples

Example 1

```
{'Back To <strong>The</strong> Future'|simpletags()}
```

The following output will be produced:

```
Back To <b>The</b> Future
```

simplify

Summary

Returns a simplified version of the input string.

Usage

```
input|simplify( [char] )
```

Parameters

Name	Type	Description	Required
char	character	The character that should be simplified.	No.

Returns

A simplified version of the input string.

Description

This operator takes a string as the input parameter. It transforms multiple consecutive characters into one. The operator can be used to remove the duplicates as it leaves only a single copy of each character. It is possible to specify only one character that should be simplified, this can be done using the optional "char" parameter. By default, the operator removes multiple spaces. Special characters must be specified using regular expression style, please refer to the table below.

Character	Description
\t	Tab (HT, TAB)
\n	Newline (LF, NL)
\r	Return (CR)
\f	Form feed (FF)
\a	Alarm / bell (BEL)
\e	Escape / think troff (ESC)

Examples

Example 1

```
{'We      don't      need      no whitespaces!'|simplify()}
```

The following output will be produced: "We don't need no whitespaces!".

Example 2

```
{'This___string__is__annoying.'|simplify( '_')}
```

The following output will be produced: "This_string_is_annoying.".

trim

Summary

Returns a stripped version of the input string.

Usage

```
input|trim( [char_list] )
```

Parameters

Name	Type	Description	Required
char_list	string	Characters that should be removed.	No.

Returns

A stripped version of the input string.

Description

This operator removes characters from the beginning and the end of the input string. By default, it will get rid of the following characters:

Character	ASCII value (dec)	ASCII value (hex)	Description
	32	0x20	An ordinary space.
\t	9	0x09	A tab.
\n	10	0x0A	A new line (line feed).
\r	13	0x0D	A carriage return.
\0	0	0x00	The NUL-byte.
\x0B	11	0x0B	A vertical tab.

Examples

Example 1

```
{'    Gizmo is not a gremlin.  '|trim()}
```

The following output will be produced: "Gizmo is not a gremlin."

upcase**Summary**

Returns a capitalized version of the input string.

Usage

```
input|upcase()
```

Returns

A capitalized version of the input string.

Description

This operator returns a capitalized version of the input string.

Examples**Example 3**

```
{'This is my string.'|upcase()}
```

The following output will be produced: "THIS IS MY STRING.".

upfirst**Summary**

Returns the input string with a capitalized initial letter.

Usage

```
input|upfirst()
```

Returns

The input string with a capitalized initial letter.

Description

This operator converts the first character of the input string to a capital letter. The resulting string is returned.

Examples**Example 1**

```
{'good bye! '|upfirst()}
```

The following string will be returned: "Good bye!".

upword**Summary**

Returns the input string with capitalized initial letters.

Usage

```
input|upword()
```

Returns

The input string with capitalized initial letters.

Description

This operator returns the input string with capitalized initial letters.

Examples**Example 3**

```
{'good bye lenin!'|upword()}
```

The following output will be produced: "Good Bye Lenin!".

wash

Summary

Returns an HTML-safe version of the input string.

Usage

```
input|wash( [type] )
```

Parameters

Name	Type	Description	Required
type	string	The type of text that should be washed.	No.

Returns

An HTML-safe version of the input string.

Description

This operator translates the input string into an HTML friendly version. It will take care of converting bogus characters to HTML-friendly replacements. The "type" parameter can be used to specify the washing type, it can be set to either "xhtml" or "email" (the default is "xhtml"). E-mail washing can be controlled using the setting of the [WashSettings] configuration block of "template.ini". All strings that may break the HTML should always be washed using this operator.

Examples

Example 1

```
{'Bogus & stuff <'|wash()}
```

The following output will be produced: "Bogus & stuff <".

Example 2

```
{'hello@example.com'|wash( 'email' )}
```

The following output will be produced:

```
hello<span class="spamfilter">SPAMFILTER</span>@example.com
```

Example 3

```
{'hello@example.com' | wash( 'email' )}
```

If a configuration override for "template.ini" exists and contains...

```
[WashSettings]
EmailDotText=[dot]
EmailAtText=[at]
```

...the following output will be produced: "hello[at]example[dot]com".

wordtoimage

Summary

Returns the input string with embedded image tags.

Usage

```
input|wordtoimage()
```

Returns

The input string with embedded image tags.

Description

This operator looks for special character sequences in the input string. When a match is found, it will be replaced by an image tag. For example, the sequence ":-)" will be replaced by a small image of a smiling face. The character sequences that should be replaced and the images that should be used are defined in the "wordtoimage.ini" configuration file.

Examples

Example 1

```
{'No problemo... :-)'}|wordtoimage()
```

This would return the input string where the ":-)" sequence would be replaced by an image tag referencing a small image of a smiling face.

wrap

Summary

Returns a wrapped version of the input string.

Usage

```
input|wrap( [ width [, break_sequence [, cut ] ] ] )
```

Parameters

Name	Type	Description	Required
width	integer	The width at which the text should be wrapped.	No.
break_sequence	string	A custom break/newline sequence.	No.
cut	boolean	TRUE (force wrap) or FALSE (do not force wrap).	No.

Returns

A wrapped version of the input string.

Description

This operator returns a wrapped version of the input string. The string will be wrapped at either the default width (80 characters) or at a width specified using the optional "width" parameter. It inserts newline characters ("\\n") or a character/sequence which is specified using the optional "break_sequence" parameter. The "cut" parameter can be set to either TRUE or FALSE - it controls whether the string should always be wrapped at the specified width or not (a word that is larger than the desired width, it will be broken apart).

Examples

Example 1

```
{'Hello world'|wrap( 5 )}
```

The following output will be produced:

Hello
world

5.7.9 URLs

exturl (page 961)

Not documented yet.

ezdesign (page 962)

Returns the input string prepended with the current design directory.

ezimage (page 963)

Returns the input string prepended with the current image directory.

ezroot (page 965)

Same as "ezurl" without "index.php" and the siteaccess name in the returned address.

ezurl (page 966)

Returns a working version of an eZ Publish URL (provided as input).

exturl

Summary

Not documented yet.

ezdesign

Summary

Returns the input string prepended with the current design directory.

Usage

```
input|ezdesign( [ quote ] )
```

Parameters

Name	Type	Description	Required
quote	string	Quote style: "no", "single" or "double" (default).	No.

Returns

The input string prepended with the current image directory.

Description

This operator returns the input string prepended with the current design directory. If the operator is unable to find the specified file within the current design, it will attempt to locate it in the fallback designs or the standard design. The "ezdesign" operator should always be used when a design related file is included in a template. It will make sure that the path to the file is always correct, regardless of the location of the eZ Publish directory, the access method, the environment, and so on.

By default, this operator returns a double-quoted string. The optional "quote" parameter can be used to control the way the address is returned: "no" (no quotes), "single" (single quotes) or "double" (double quotes, the default). Dropping quotes is useful when specifying CSS files in the following way:

```
<style type="text/css">
    @import url({'stylesheets/core.css'|ezdesign( 'no' )});
    @import url({'stylesheets/ezmain.css'|ezdesign( 'no' )});
    @import url({'stylesheets/ezsystems.css'|ezdesign( 'no' )});
</style>
```

ezimage

Summary

Returns the input string prepended with the current image directory.

Usage

```
input|ezimage( [ quote [, slash_skip ] ] )
```

Parameters

Name	Type	Description	Required
quote	string	Quote style: "no", "single" or "double" (default).	No.
skip_slash	boolean	Include (FALSE, default) or skip (TRUE) the first slash.	No.

Returns

The input string prepended with the current image directory.

Description

This operator prepends the input string with the location of the image directory used by the current design. If the operator is unable to find the specified file within the "images" subdirectory of the current design, it will attempt to locate it in the "images" subdirectory of the fallback designs or the standard design. The "ezimage" operator should always be used when an image is included in a template. It will make sure that the path to the image is always correct, regardless of the location of the eZ Publish directory, the access method, the environment, and so on.

By default, this operator returns a double-quoted string. The optional "quote" parameter can be used to control the way the address is returned: "no" (no quotes), "single" (single quotes) or "double" (double quotes, the default). The optional "skip_slash" parameter can be used to get rid of the first slash within the string that is being returned (when set to false()).

Examples

Example 1

In this example, the design "my_company" is used by the siteaccess. Images should be included in the following way:

```
<img src={'banner.jpg'|ezimage()} alt="My banner" ...>
```

The following output will be produced:

```

```

If eZ Publish is unable to find the image within the images directory of the current design directory, it will attempt to find it within the images subdirectory of the additional designs. At last, it will fallback to the standard design. In this case, the output will be the following:

```

```

ezroot**Summary**

Same as "ezurl" without "index.php" and the siteaccess name in the returned address.

Usage

```
input|ezroot( [ quote [, type ] ] )
```

Parameters

Name	Type	Description	Required
quote	string	Quote style: "no", "single" or "double" (default).	No.
type	string	URL type: "full" or "relative" (default).	No.

Returns

A string containing a working version of the input address.

Description

This operator works almost in the same way as the "ezurl" (page 966) operator. The only difference is that it does not include "index.php" or the name of the siteaccess in the returned address. In other words, it returns an address which points to the root of the eZ Publish directory.

ezurl

Summary

Returns a working version of an eZ Publish URL (provided as input).

Usage

```
input|ezurl( [ quote [, type ] ] )
```

Parameters

Name	Type	Description	Required
quote	string	Quote style: "no", "single" or "double" (default).	No.
type	string	URL type: "full" or "relative" (default).	No.

Returns

A quoted string containing a valid / working version of the input URL.

Description

This operator takes an eZ Publish URL as input (either a system URL or a virtual URL); based on the location of the eZ Publish folder, the access settings and the environment, it will produce a valid address. All eZ Publish URLs that are specified in templates should always be piped through this operator; it will make sure that the URLs work regardless where eZ Publish is installed, which access method is used, and so on.

By default, this operator returns a relative URL as a double-quoted string. The optional "quote" parameter can be used to control the way the address is returned: "no" (no quotes), "single" (single quotes) or "double" (double quotes, the default). The optional "type" parameter controls whether relative or full URL is returned.

Examples

Example 1

Let's say that we're running a site called "my_company" (name of the siteaccess) and that we wish to create a link to the full view of node number 1024. Instead of specifying the entire URL (domain and all included) in the link tag, we pipe "/content/view/full/1024" or the virtual URL (for example "/test") through the "ezurl" operator:

```
<a href={'/content/view/full/1024'|ezurl('full')}>Test</a>
<a href={'/test'|ezurl('full')}>Test</a>
```

The operator will take care of translating the URLs into valid addresses depending on the setup and the environment eZ Publish is running in. If eZ Publish is running in a virtual host environment (page 70) and uses the host access method, the following type of URLs will be produced:

```
”http://www.example.com/content/view/full/1024”
”http://www.example.com/test”
```

The "index.php" part of the URL will be suppressed by the virtual host configuration (page 73). The name of the siteaccess will not appear in the URL because eZ Publish will use the domain/host to figure out which siteaccess to use.

If eZ Publish is running in a non-virtual host environment and uses the uri access method, the following URLs will be produced:

```
”http://www.example.com/index.php/my_company/content/view/full/1024”
”http://www.example.com/index.php/my_company/test”
```

Example 2

```
<a href={'/content/view/full/1024'|ezurl()}>Test</a>
<a href={'/test'|ezurl()}>Test</a>
```

If eZ Publish is running in a virtual host environment and uses the host access method, the following type of URLs will be produced:

```
”/content/view/full/1024”
”/test”
```

The "index.php" part of the URL will be suppressed by the virtual host configuration. The name of the siteaccess will not appear in the URL because eZ Publish will use the domain/host to figure out which siteaccess to use.

If eZ Publish is running in a non-virtual host environment and uses the uri access method, the following URLs will be produced:

```
”/index.php/my_company/content/view/full/1024”
”/index.php/my_company/test”
```

5.7.10 Variable and type handling

count (page 969)

Returns the count of the input parameter.

float (page 970)

Converts the input parameter to a float.

get_class (page 971)

Returns the class name of an object.

get_type (page 973)

Returns the type of the provided variable.

int (page 975)

Converts the input parameter to an integer.

is_array (page 976)

Returns TRUE if the provided variable is an array.

is_boolean (page 978)

Returns TRUE if the provided variable is a boolean.

is_class (page 980)

Returns TRUE if an object is an instance of a specific class.

is_float (page 982)

Returns TRUE if the provided variable is a float.

is_integer (page 984)

Returns TRUE if the provided variable is an integer.

is_null (page 986)

Returns TRUE if the provided variable is NULL.

is_numeric (page 988)

Returns TRUE if the provided variable is a number.

is_object (page 990)

Returns TRUE if the target variable is an object.

is_set (page 992)

Returns TRUE if the value of the provided variable is set.

is_string (page 994)

Returns TRUE if the provided variable is a string.

is_unset (page 996)

Returns TRUE if the provided variable is not set (has no value).

count**Summary**

Returns the count of the input parameter.

Usage

```
input|count()
```

Returns

An integer revealing the count.

Description

This operator returns the count of the input parameter. The following table shows how the operator works with different variable types.

Type	Description
Array	The number of elements is returned.
Object	The number of object attributes is returned.
String	The length of the string is returned.
Number	The value itself is returned.
Boolean	FALSE results in 0 and TRUE results in 1.
Other	0 is returned.

Examples**Example 1**

```
{array( 1, 2, 5 )|count()}
```

The following output will be produced: "3".

float**Summary**

Converts the input parameter to a float.

Usage

```
input|float()
```

Returns

A float representation of the input parameter.

Description

This operator attempts to convert the input parameter to a float. It returns the converted value as a float. If the operator is unable to do the conversion, it will return a value of zero (0).

Examples**Example 1**

```
{def $pi='3.1415'}  
Value: {$pi|float()}
```

Converts the string "3.1415" and returns it as a float. The following output will be produced: "Value: 3.1415".

Example 2

```
{def $pi='three point fourteen'}  
Value: {$pi|float()}
```

The following output will be produced: "Value: 0".

get_class

Summary

Returns the class name of an object.

Usage

```
input|get_class( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

A string containing the class name or FALSE.

Description

This operator gets the class of the input parameter or the target variable. It returns the PHP class name as a string. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated. If the provided variable is not an object then the operator will return FALSE.

Examples

Example 1

```
{def $my_variable="Test"}  
  
{if get_class( $my_variable )}  
    Class detected.  
{else}  
    There is no class.  
{/if}
```

The following output will be produced: "There is no class.".

Example 2

```
{get_class( $node )}
```

If \$node is an actual content node, the following output will be produced: "ezcontentobjecttree-node".

get_type

Summary

Returns the type of the provided variable.

Usage

```
input|get_type( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

A string containing the type of the provided variable.

Description

This operator can be used to check the type of a variable. It returns the type of the input parameter or the target variable as a string. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

- If the data is an object, the string "object" and the name of the class will be returned.
- If the data is an array, the string "array" and the number of elements will be returned.
- If the data is a string, the string "string" and the length of the string will be returned.

Examples

Example 1

```
{def $my_variable='ich bin'}
{$my_variable|get_type()}
```

The following output will be produced: "string".

Example 2

```
{def $my_variable='ich bin'  
    $your_variable=array( 'du', 'bist' )}  
  
{$my_variable|get_type( $your_variable )}
```

The following output will be produced: "array[2]".

int**Summary**

Converts the input parameter to an integer.

Usage

```
input|int()
```

Returns

An integer representation of the input parameter.

Description

This operator attempts to convert the input parameter to an integer. It will return the converted value as an integer. If the operator is unable to do the conversion, it will return the value of zero (0).

Examples**Example 1**

```
{def $number='57'}  
Value: {$number|int()}
```

Converts the string "57" and returns it as an integer. The following output will be produced: "Value: 57".

Example 2

```
{def $number='fiftyseven'}  
Value: {$number|int()}
```

The following output will be produced: "Value: 0".

is_array

Summary

Returns TRUE if the provided variable is an array.

Usage

```
input|is_array( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is an array. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=array( 1, 2, 3 )}

{if $my_variable|is_array()}
    It is an array.
{else}
    It is not an array.
{/if}
```

The following output will be produced: "It is an array.".

Example 2

```
{def $my_variable=array( 1, 2, 3 )}

{if is_array( $my_variable )}
    It is an array.
{else}
    It is not an array.
{/if}
```

The following output will be produced: "It is an array".

Example 3

```
{def $a=array( 1, 2, 3 )
    $b='Mobile instrument.'}

{if $a|is_array( $b )}
    It is an array.
{else}
    It is not an array.
{/if}
```

The following output will be produced: "It is not an array".

is_boolean

Summary

Returns TRUE if the provided variable is a boolean.

Usage

```
input|is_boolean( target )
```

Parameters

Name	Type	Description	Required
target	any	Target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is a boolean. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=true
{if $my_variable|is_boolean()
    It is a boolean.
{else}
    It is not a boolean.
{/if}}
```

The following output will be produced: "It is a boolean.".

Example 2

```
{def $my_variable=true
{if is_boolean( $my_variable )}
    It is a boolean.
{else}
    It is not a boolean.
{/if}
```

The following output will be produced: "It is a boolean.".

Example 3

```
{def $my_variable=true
    $your_variable='BOFID'}
{if $my_variable|is_boolean( $your_variable )}
    It is a boolean.
{else}
    It is not a boolean.
{/if}
```

The following output will be produced: "It is not a boolean.".

is_class

Summary

Returns TRUE if an object is an instance of a specific class.

Usage

```
input|is_class( name [, object ] )
```

Parameters

Name	Type	Description	Required
name	string	The class name that should be matched.	Yes.
object	object	An object of any type (instead of the input parameter).	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator attempts to find out the class name of the object that is provided either as the input parameter or the "object" parameter. If the name of the class matches the name provided using the "name" parameter, the function will return TRUE; otherwise FALSE will be returned.

Examples

Example 1

```
{if $node|is_class( 'ezcontentobjecttreenode' )}  
Everything is okay.  
{else}  
Something is wrong.  
{/if}
```

As long as the \$node refers to an instance of the "ezcontentobjecttreenode" class, the following output will be produced: "Everything is okay.".

Example 2

```
{if is_class( 'ezcontentobjecttreenode', $node )}  
Everything is okay.  
{else}  
Something is wrong.  
{/if}
```

As long as the \$node refers to an instance of the "ezcontentobjecttreenode" class, the following output will be produced: "Everything is okay".

is_float

Summary

Returns TRUE if the provided variable is a float.

Usage

```
input|is_float( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

true or false, see description for details.

Description

This operator checks if the input parameter or the target variable is a float. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=3.1415}

{if $my_variable|is_float()}
    It is a float.
{else}
    It is not a float.
{/if}
```

The following output will be produced: "It is a float.".

Example 2

```
{def $my_variable=3.1415}

{if is_float( $my_variable )}
    It is a float.
{else}
    It is not a float.
{/if}
```

The following output will be produced: "It is a float.".

Example 3

```
{def $a=3.1415
    $b='Mobile instrument.'}

{if $a|is_float( $b )}
    It is a float.
{else}
    It is not a float.
{/if}
```

The following output will be produced: "It is not a float.".

is_integer

Summary

Returns TRUE if the provided variable is an integer.

Usage

```
input|is_integer( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is an integer. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=3}

{if $my_variable|is_float()}
    It is an integer.
{else}
    It is not an integer.
{/if}
```

The following output will be produced: "It is an integer.".

Example 2

```
{def $my_variable=3}

{if is_float( $my_variable )}
    It is an integer.
{else}
    It is not an integer.
{/if}
```

The following output will be produced: "It is an integer".

Example 3

```
{def $a=3
    $b='Mobile instrument.'}

{if $a|is_float( $b )}
    It is an integer.
{else}
    It is not an integer.
{/if}
```

The following output will be produced: "It is not an integer".

is_null

Summary

Returns TRUE if the provided variable is NULL.

Usage

```
input|is_null( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is NULL. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated. Please note that an integer with a value of zero is not the same as NULL.

Examples

Example 1

```
{def $my_variable=3}

{if $my_variable|is_null()}
    It is NULL.
{else}
    It is not NULL.
{/if}
```

The following output will be produced: "It is not NULL.".

Example 2

```
{def $my_variable=3}

{if is_null( $my_variable )}
    It is NULL.
{else}
    It is not NULL.
{/if}
```

The following output will be produced: "It is not NULL.".

Example 3

```
{def $a=3
    $b='Mobile instrument.'}

{if $a|is_null( $b )}
    It is NULL.
{else}
    It is not NULL.
{/if}
```

The following output will be produced: "It is not NULL.".

is_numeric

Summary

Returns TRUE if the provided variable is a number.

Usage

```
input|is_numeric( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is a number or a numeric string (a string containing a number). If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=3}

{if $my_variable|is_numeric()}
    It is a number.
{else}
    It is not a number.
{/if}
```

The following output will be produced: "It is a number.".

Example 2

```
{def $my_variable='256'}

{if is_numeric( $my_variable )}
    It is a number.
{else}
    It is not a number.
{/if}
```

The following output will be produced: "It is a number.".

Example 3

```
{def $a=3
    $b='Mobile instrument.'}

{if $a|is_numeric( $b )}
    It is a number.
{else}
    It is not a number.
{/if}
```

The following output will be produced: "It is not a number.".

is_object

Summary

Returns TRUE if the target variable is an object.

Usage

```
input|is_object( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is an object (as opposed to simple types like integer, string, etc.). If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable=3}

{if $my_variable|is_object()}
    It is an object.
{else}
    It is not an object.
{/if}
```

The following output will be produced: "It is not an object.".

Example 2

```
{def $my_variable='256'}

{if is_object( $my_variable )}
    It is an object.
{else}
    It is not an object.
{/if}
```

The following output will be produced: "It is not object.".

Example 3

```
{def $a=3
    $b='Mobile instrument.'}

{if $a|is_object( $b )}
    It is an object.
{else}
    It is not an object.
{/if}
```

The following output will be produced: "It is not an object.".

is_set

Summary

Returns TRUE if the value of the provided variable is set.

Usage

```
is_set( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Yes.

Returns

TRUE or FALSE.

Description

This operator checks if the value of the target parameter is a non-false value (meaning that it is set). If it is, the operator will return TRUE, otherwise FALSE will be returned. Please note that this operator does not take an input parameter.

Examples

Example 1

```
{if is_set( $whatever )}
    It is set.
{else}
    It is not set.
{/if}
```

The following output will be produced: "It is not set." - because \$whatever is not declared and/or defined.

Example 2

```
{def $whatever='We need more rocket fuel!'}
{if is_set( $whatever )}
    It is set.
{else}
    It is not set.
{/if}
```

The following output will be produced: "It is set.".

is_string

Summary

Returns TRUE if the provided variable is a string.

Usage

```
input|is_string( target )
```

Parameters

Name	Type	Description	Required
target	any	The target variable.	Only if the input parameter is omitted.

Returns

TRUE or FALSE.

Description

This operator checks if the input parameter or the target variable is a string. If it is, the operator will return TRUE, otherwise FALSE will be returned. If both the input parameter and the target variable are provided, it is the target variable that will be evaluated.

Examples

Example 1

```
{def $my_variable='Commodore'}

{if $my_variable|is_string()}
    It is a string.
{else}
    It is not a string.
{/if}
```

The following output will be produced: "It is a string.".

Example 2

```
{def $my_variable='Amiga'}

{if is_string( $my_variable )}
    It is a string.
{else}
    It is not a string.
{/if}
```

The following output will be produced: "It is a string.".

Example 3

```
{def $a='C64',
    $b=128}

{if $a|is_string( $b )}
    It is a string.
{else}
    It is not a string.
{/if}
```

The following output will be produced: "It is not a string.".

is_unset

Summary

Returns TRUE if the provided variable is not set (has no value).

Usage

```
is_unset( target )
```

Parameters

Name	Type	Description	Required
test	any	The target variable.	Yes.

Returns

TRUE or FALSE.

Description

This operator checks if the target variable is set (has a value). If it is, the operator returns FALSE, otherwise TRUE will be returned.

Examples

Example 1

```
{def $my_variable=true()
{if is_unset( $my_variable )}
    Yes.
{else}
    No.
{/if}}
```

The following output will be produced: "No.".

Example 2

```
{def $my_variable=false()
{is_unset( $my_variable )}
    Yes.
{else}
    No.
{/if}}
```

The following output will be produced: "Yes.".

5.8 Template functions

The template functions are documented in the following sections:

- Debugging (page [999](#))
- Miscellaneous (page [1004](#))
- Variables (page [1017](#))
- Visualization (page [1030](#))

5.8.1 Debugging

debug-accumulator (page 1000)

Generates debug statistics for a block of template code.

debug-timing-point (page 1002)

Measures the time it takes to process a block of template code.

debug-trace (page 1003)

Generates an XDebug dump which can be traced/analyzed.

debug-accumulator

Summary

Generates debug statistics for a block of template code.

Usage

```
{debug-accumulator [ id=id ] [ name=name ]}  
...  
{/debug-accumulator}
```

Parameters

Name	Type	Description	Required
id	string	A unique identifier string for an accumulator.	No.
name	string	A name that will be used in the debug output.	No.

Description

This mechanism generates some debug statistics based on the code that is encapsulated by ”{debug-accumulator}” and ”{/debug-accumulator}”. The encapsulated code will be processed normally. The number of calls, total time and average time will be shown in the debug output.

The ”id” and the ”name” parameters are optional. The ”id” parameter can be used to uniquely identify one accumulator. This means that if the same accumulator is used at multiple locations, the values will be accumulated and appended. The ”name” parameter can be used to assign a name which will be used in the debug output.

Examples

Example 1

```
{debug-accumulator}  
  
{def $nodes=fetch( 'content', 'tree', hash( 'parent_node_id', 2 ) )}  
  
{foreach $nodes as $node}  
    {$node.name|wash()}  
{/foreach}  
  
{/debug-accumulator}
```

This example demonstrates how the "debug-accumulator" mechanism can be used to generate some debug statistics based on the encapsulated template code.

debug-timing-point

Summary

Measures the time it takes to process a block of template code.

Usage

```
{debug-timing-point [ id=id ]}  
...  
{/debug-timing-point}
```

Parameters

Name	Type	Description	Required
id	string	An identification string for the timing point.	No.

Description

This mechanism starts a timer, executes the template code that is encapsulated by ”{debug-timing-point...}” and ”{/debug-timing-point}”, finally it stops the timer. It can be used to figure out how much time it takes to process a block of template code and/or to isolate debug messages that are generated between the timing points. The measurements will show up in the debug message. The optional ”id” parameter can be used to assign an identification string to the block.

Examples

Example 1

```
{debug-timing-point id='test'}  
  
{def $nodes=fetch( 'content', 'tree', hash( 'parent_node_id', 2 ) )}  
  
{foreach( $nodes as $node )}  
    {$node.name|wash()  
{/foreach}  
  
{/debug-timing-point}
```

This example demonstrates how the ”debug-timing-point” mechanism can be used to measure the amount of time it takes to fetch and print the names of all nodes that are below node number 2.

debug-trace

Summary

Generates an XDebug dump which can be traced/analyzed.

Usage

```
{debug-trace [ id=id ]}  
...  
{/debug}
```

Parameters

Name	Type	Description	Required
id	string	The name of the debug file.	No.

Description

This mechanism makes it possible to trace a block of code using "XDebug". The result will be a trace file made by XDebug which can be analyzed. If XDebug is not installed and enabled, this mechanism will not do anything. The "id" parameter can be used to name the trace file. The file extension will be ".xt". The default ID/name is "template-debug". Please note that the trace file will be reset every time a debug-trace is encountered; it is a good idea to have one unique ID per entry.

Examples

Example 1

```
{debug-trace id='fetch-trace'}  
  
{def $nodes=fetch( 'content', 'tree', hash( 'parent_node_id', 2 ) )}  
  
{foreach( $nodes as $node )}  
    {$node.name|wash()}  
{/foreach}  
  
{/debug-trace}
```

This will generate an XDebug trace file called "fetch-trace.xt".

5.8.2 Miscellaneous

cache-block (page 1005)

Caches the contents of a template block.

fetch alias (page 1009)

Executes a fetch based on configuration settings.

include (page 1012)

Includes a file.

ldelim (page 1013)

Outputs a left curly bracket, "{".

literal (page 1014)

Instructs the parser to ignore a block of template code.

rdelim (page 1015)

Outputs a right curly bracket, "}".

run-once (page 1016)

Assures that a block of template code is run only once within a page view.

cache-block

Summary

Caches the contents of a template block.

Usage

```
{cache-block [ keys=keys ]  
            [ expiry=expiry ]  
            [ ignore_content_expiry ]  
            [ subtree_expiry=subtree_expiry ]}  
  
...  
  
{/cache-block}
```

Parameters

Name	Type	Description	Required
keys	string or array	Cache key(s) - either as a string or an array of strings.	No.
expiry	integer	The number of seconds that the cache should be allowed to live.	No.
ignore_content_expiry	-	Disables cache expiry when new content is published.	No.
subtree_expiry	string	A subtree that expires the cache block.	No.

Description

This solution makes it possible to reduce the processing time of the main template ("pagelayout.tpl"), which often contains a lot of dynamic elements. It can be used to instruct the system to store and reuse cached blocks of template code based on different conditions.

A typical example of where the "cache-block" mechanism should be used is the main menu of a site. The menu is often dynamically generated by fetching and displaying information about some nodes. It is usually the same for almost every page, therefore it should not be generated from scratch every time eZ Publish is instructed to render a page. This is where the "cache-block" solution comes in. In this particular scenario, it can be used to cache the contents of the main menu and thus reduce the processing time for each page load.

Cache keys

The "keys" parameter can be used to define the uniqueness of a cache block. It must be either a string or an array of strings. By default, eZ Publish uses the name of the template and the position of the cache block as keys. This means that if the cache block is common for all cases that use the given template (normally "pagelayout.tpl"), there is no need to define any keys. However, the "keys" parameter is quite handy when it comes to relating a cache block to something specific (for example URLs, users, etc.). Please refer to the examples below for a demonstration of how this parameter can be used.

Time based expiration

The "expiry" parameter makes it possible to manually specify how long a cache block should live (number of seconds). The default expiration time is two hours (this is hardcoded in the system and can not be configured). If an object is published, all blocks will automatically be expired. A value of zero will produce a cache block that will never expire.

Content expiration

By default, all cache blocks will be expired whenever an object is published. If the "ignore_content_expiry" parameter is used, the cache block will not be expired when an object is published. However, it will still expire after two hours unless an alternative time is specified using the "expiry" parameter.

Subtree expiration

The "subtree_expiry" parameter can be used to bind the expiration of a cache block to a certain part of the content node tree. When this is done, the block will expire if an object is published below the given subtree instead of the entire tree. In addition, it will also expire after two hours unless an alternative time is specified using the "expiry" parameter.

Tips and tricks

Since cache blocks themselves also produce some overhead, too many blocks may lead to longer response times than expected. Because of this, only a few cache blocks should be used; and their keys should be as unique as possible. It is often very efficient to have two large cache blocks. One which caches all header information (title, path, etc.) and one which will take care of the bottom/footer of the page. This solution combined with a nested cache block used for the main menu (or several menus, etc.) often leads to good results. Please note that although the cache block mechanism was designed to minimize the processing of the main template, it may also be used in view templates. For example, it is possible to cache a part of a view - this is typically useful when the viewcache is frequently deleted. Another scenario is when the view cache is turned off and there is a need to create a cache on a per-user basis.

Examples

Example 1

```
...
{include uri='design:page_toppath.tpl'}

{cache-block}
    {include uri='design:menu.tpl'}
{/cache-block}

{$module_result.content}

{include uri='design:page_bottom.tpl'}
...
```

This example demonstrates how the cache block solution can be used to cache the contents of a menu (which will be the same for all pages) in the pagelayout.

Example 2

```
{cache-block expiry=130}
...
{/cache-block}
```

This example demonstrates how to create a cache block that will expire after 130 seconds.

Example 3

```
{cache-block keys=$uri_string}
...
{/cache-block}
```

This example demonstrates how to create a cache block that will be unique for every URL.

Example 4

```
{cache-block keys=array( $uri_string, $current_user.contentobject_id )}
...
{/cache-block}
```

This example demonstrates how to create a cache block that is unique for each URL and each user.

Example 5

```
{cache-block ignore_content_expiry}  
...  
{/cache-block}
```

This example demonstrates how to create a cache block that will not expire when new content is published. However, it will expire every second hour unless an alternative "time to live" value is specified using the "expiry" parameter.

Example 6

```
{cache-block subtree_expiry='products/'}  
...  
{/cache-block}
```

This example demonstrates how to create a cache block that will expire only if something is modified within the "products/" subtree, for example if a product is modified or a new product is published.

fetch_alias**Summary**

Executes a fetch based on configuration settings.

Usage

```
fetch_alias( alias_name, hash( [ parameter1, value1,
                               [ parameter2, value2 ] ] ) )
```

Parameters

Name	Type	Description	Required
alias_name	string	The name of the fetch alias that should be used.	Yes.
parameter1	string	The name of parameter 1.	No.
value1	string	The value of parameter 1.	No.
parameter2	string	The name of parameter2.	No.
value2	string	The value of parameter 2.	No.

Description

This function can be thought of as a configuration-file based version of the "fetch" (page 813) operator. It makes it possible to move data fetching blocks from template code to a configuration file and thus gather all fetches at one place. The advantage of such a scenario is that it allows quick modifications without the need of having to locate and modify different templates. The fetch aliases must be defined in a configuration override for "fetchalias.ini". Each fetch has to be defined within its own block with a unique name. The following code shows the basic syntax/structure of a fetch block.

```
[fetch_alias_name]
Module=module_name
FunctionName=function_name
Parameter[parameter1]=fetch_alias_name1
Parameter[parameter2]=fetch_alias_name2
...
Constant [parameter3]=<any value>
Constant [parameter4]=<any value>
...
```

Directive	Description
Module	The name of the target module (for example

	”content”).
FunctionName	The name of the target fetch function (for example ”list”).
Parameter	The ”Parameters” array may be used to specify variables that will be set in the template(s). The ”parameter_name” maps to the parameter name used in normal fetch functions. The ”fetch_alias_name” will be the parameter name used in the template(s).
Constant	Parameters that are defined as constants within the regular fetch function(s).

Examples

Example 1

Configuration block:

```
[object]
Module=content
FunctionName=object
Parameter[object_id]=id
```

Template code:

```
{def $object=fetch_alias( 'object', hash( 'id', 1 ) )}
```

This example demonstrates how to fetch an object.

Example 2

Configuration block:

```
[comments]
Module=content
FunctionName=list
Constant[sort_by]=published;0
Parameter[parent_node_id]=parent_node_id
Constant[class_filter_type]=include
Constant[class_filter_array]=comment
```

Template code:

```
{def $comments=fetch_alias( 'comments', hash( 'parent_node_id', 42 ) )}
```

This example demonstrates how to fetch comments.

Example 3

Configuration block:

```
[news_list]
Module=content
FunctionName=tree
Constant[sort_by]=published;0
Constant[class_id]=2
Constant[parent_node_id]=2
Constant[class_filter_type]=include
Constant[limit]=10
Constant[class_filter_array]=2
```

Template code:

```
{foreach fetch_alias( 'news_list' ) as $article}
    {node_view_gui node=$article}
{/foreach}
```

This example demonstrates how to fetch and display the 10 latest news articles using full view.

include

Summary

Includes a file.

Usage

```
{include uri='path_to_file' [ name='namespace' ] [ parameter(s)='value(s)' ]}
```

Parameters

Name	Type	Description	Required
uri	string	Path + name of the file that should be included.	Yes.
name	string	Alternative namespace for the included template.	No.
other parameters	any	Parameters that will be passed to the included template.	No.

Description

This function includes a file in the template from where the function was called. The "uri" parameter must be used to specify the target file. In most cases, the value of this parameter starts with a "design:", which tells the system to look for the desired template within the current (and fallback) design resources. The "name" parameter can be used to specify an alternative namespace for the included template, this is useful for avoiding variable name clashes when including other templates. All other parameters will be passed to the included template as template variables. This function makes it possible to share template code among different parts of the solution.

Examples

Example 1

```
{include uri='design:example/menu.tpl' something='Hello world'}
```

This example demonstrates how to include a template called "menu.tpl" (which is located within the "example" subdirectory of the "templates" directory. If eZ Publish is unable to find the template in the current design, it will automatically attempt to locate it in one of the fallback designs or the standard design. The value of the "something" parameter will be available through a variable called \$something within the template that is included.

ldelim**Summary**

Outputs a left curly bracket, "{'".

Usage

```
{ldelim}
```

Description

This function displays a left curly bracket, "{'". It can for example be used to add JavaScript functions in a template.

Examples**Example 1**

```
<script language="JavaScript" type="text/javascript">
<!--
function foo()
{ldelim}
    alert ('Call me!');
{rdelim}
//-->
</script>
```

This example demonstrates how to use the "ldelim" and "rdelim" template functions to generate curly brackets.

literal**Summary**

Instructs the parser to ignore a block of template code.

Usage

```
{literal}  
...  
{/literal}
```

Description

This function can be used to encapsulate foreign code (for example JavaScript) that makes use of characters that may confuse the template parser. Everything that is inside a literal block will be completely ignored by the parser.

Examples**Example 1**

```
{literal}  
<script language="JavaScript" type="text/javascript">  
<!--  
function foo()  
{  
    alert ( "Call me" );  
}  
//-->  
</script>  
{/literal}
```

This example demonstrates how to include a JavaScript snippet in a template using the "{literal}" and "{/literal}" notation.

rdelim**Summary**

Outputs a right curly bracket, "}".

Usage

```
{rdelim}
```

Description

This function displays a right curly bracket, "}". It can for example be used to add JavaScript functions in a template.

Examples**Example 1**

```
<script language="JavaScript" type="text/javascript">
<!--
function foo()
{ldelim}
    alert ('Call me!');
{rdelim}
//-->
</script>
```

This example demonstrates how to use the "ldelim" and "rdelim" template functions to generate curly brackets.

run-once**Summary**

Assures that a block of template code is run only once within a page view.

Usage

```
{run-once}
  ...
{/run-once}
```

Description

This function makes sure that a block of template code is processed only once within a page view. It is typically useful when it comes to displaying elements that should appear once or to do timeconsuming calculations that only has to be processed once (and the result is included in multiple templates).

Examples**Example 1**

```
{def $elements=array( 'A', 'B', 'C' )}

{foreach $elements as $element}

  {run-once}
    Hello world <br/>
 {/run-once}

  {$element} <br />

{/foreach}
```

The following output will be produced:

```
Hello World
A
B
C
```

5.8.3 Variables

append-block (page 1018)

Redirects the output from multiple blocks of template code to an array.

def (page 1020)

Declares (and defines) a variable. Warns if the variable already exists.

default (page 1022)

Deprecated.

let (page 1023)

Deprecated.

sequence (page 1024)

Creates a sequence that can be iterated.

set (page 1025)

Sets the value of a variable.

set-block (page 1027)

Redirects the output from a block of template code to a string.

undef (page 1029)

Destroys previously defined variable(s).

append-block

Summary

Redirects the output from multiple blocks of template code to an array.

Usage

```
{append-block variable=$variable [ name=name ] [ scope=scope ]}
...
{/append-block}
```

Parameters

Name	Type	Description	Required
name	string	Name of the namespace.	No.
scope	string	The scope ("global", "root" or "relative").	No.
variable	string	The name of variable that will be returned.	Yes.

Description

This mechanism will silently process all template code which is encapsulated by "{append-block ...}" and "{/append-block}". It will not produce any actual output. Instead, the generated output will be assigned to a variable specified by the "variable" parameter (as an array). If the variable does not exist, it will be automatically created. If the same target variable is used in several blocks, the function will simply add new elements to the array and thus the previous contents will be preserved.

Examples

Example 1

```
{append-block variable=$alien}
    It seems to have a life,
{/append-block}

...
{append-block variable=$alien}
    organic life...
{/append-block}
```

```
...
{foreach $alien as $element}
    {$element} <br />
{/foreach}
```

This example demonstrates how to create an array called \$alien using the "append-block" mechanism. The output from the code that is encapsulated by "{append-block ...}" and "{/append-block}" will be assigned as elements to the target variable. When the \$alien array is inspected, the following output will be produced:

It seems to have a life,
organic life...

def**Summary**

Declares (and defines) a variable. Warns if the variable already exists.

Usage

```
{def $var1=value1 [ $var2=value2 [...] ]}
```

Parameters

Name	Type	Description	Required
\$var1	string	Name of variable number one (with a dollar sign in front of it).	Yes.
value1	any	The value that should be assigned to variable one.	Yes.
\$var2	string	Name of variable number two (with a dollar sign in front of it).	No.
value2	any	The value that should be assigned to variable two.	No.

Description

This function allows the declaration and definition of a single variable or a group of variables. The "undef" (page 1029) function can be used to flush/destroy variables that were created using the "def" function. Please note that this function does not support the "name" and the "scope" parameters (like the old {let} did).

Replacement for "default"

The following technique can be used as a replacement for the old "default" function:

```
{if is_set( $a )|not}
  {def $a=5}
```

Examples**Example 1**

```
{def $oranges=13}
```

This example demonstrates how the "def" function can be used to declare a variable called "oranges". The variable will be declared as an integer with a value of 13.

Example 2

```
{def $oranges=13 $apples='There are no apples.'}
```

or

```
{def $oranges=13  
$apples='There are no apples.'}
```

These code snippets demonstrate how the "def" function can be used to declare multiple variables. A variable called "oranges" will be declared as an integer with a value of 13. A variable called "apples" will be declared as a string containing the following characters: "There are no apples.".

default**Summary**

Deprecated.

Description

This function is deprecated and should not be used. If you need more information, please refer to the [documentation of the old template syntax](#) (it is still present in some of the default templates included in the distributions).

let

Summary

Deprecated.

Description

This function is deprecated and should not be used. If you need more information, please refer to the [documentation of the old template syntax](#) (it is still present in some of the default templates included in the distributions).

sequence**Summary**

Creates a sequence that can be iterated.

Usage

```
{sequence name=name loop=loop}
```

Parameters

Name	Type	Description	Required
name	string	The name of the target namespace.	Yes.
loop	array	The iteration elements (as an array).	Yes.

Description

This function allows the creation of a sequence which can be iterated. If the number of iterations exceed the length of the sequence, the contents of the sequence will be wrapped and thus repeated. It is typically useful when it comes to the creation of lists / tables with alternating colors. Both the name and the elements (which will be iterated) must be defined.

set

Summary

Sets the value of a variable.

Usage

```
{set $var1=value1 [ var2=value2 [...] ] [ name=name ] [ scope=scope ]}
```

Parameters

Name	Type	Description	Required
var1	string	Name of variable number one (with a dollar sign in front of it).	Yes.
value1	any	The value that should be assigned to variable 1.	Yes.
var2	string	Name of variable number two (with a dollar sign in front of it).	No.
value2	any	The value that should be assigned to variable 2.	No.
name	string	The name of the target namespace.	No.
scope	string	The scope ("global", "root" or "relative").	No.

Description

This function makes it possible to assign new values to variables that previously have been declared using either the "def" (page 1020) function. The "name" and "scope" parameters are optional and can be used to set the desired namespace and scope.

Examples

Example 1

```
{def $apples=4}
Before: {$apples} <br/>
...
{set $apples=8}
After: {$apples} <br/>
```

The following output will be produced:

Before: 4

After: 8

Example 1

```
{def name=ns1 $var1='ns1 org value'}
{def name=ns2 $var1='ns2 org value'}

Original values: <br/>
$ns1:var1 : {$ns1:var1} <br/>
$ns1:ns2:var1 : {$ns1:ns2:var1} <br/>
...
{set name=ns1 scope=root var1='new value'}
{set var1='new value'}
...
New values: <br />
$ns1:var1 : {$ns1:var1}<br/>
$ns1:ns2:var1 : {$ns1:ns2:var1}<br/>
```

The following output will be produced:

```
Original values:
$ns1:var1 : ns1 org value
$ns1:ns2:var1 : ns2 org value

New values:
$ns1:var1 : new value
$ns1:ns2:var1 : new value
```

set-block

Summary

Redirects the output from a block of template code to a string.

Usage

```
{set-block variable=$variable [ name=name ] [ scope=scope ]}
...
{/set-block}
```

Parameters

Name	Type	Description	Required
variable	string	The name of the variable (with dollar sign).	Yes.
name	string	The name of the target namespace.	No.
scope	string	The scope ("global", "root" or "relative").	No.

Description

This mechanism will silently process all template code which is encapsulated by "{set-block ...}" and "{/set-block}". It will not produce any actual output. Instead, the generated output will be assigned to a variable specified by the "variable" parameter (as a string). If the variable does not exist, it will be automatically created. If the same target variable is used in several blocks, its contents will be overwritten every time a new block is processed.

Examples

Example 1

```
{set-block variable=$example}
    {def $test=array( 'x', 'y', 'z' )}
    Hello world - {$test[1]}
{/set-block}
...
{$example}
```

The code which is encapsulated by "set-block" will not produce any output. Instead, the output will be put into a string called \$example. When this variable is accessed directly, the following output will be produced: "Hello world - y".

Example 2

```
{set-block scope=root variable=cache_ttl}0{/set-block}
```

This will put zero into the "cache_ttl" global variable and thus disable view caching for this page.

undef

Summary

Destroys previously defined variable(s).

Usage

```
{undef [ $var1 [ ... ] ]}
```

Parameters

Name	Type	Description	Required
var1	string	The name of the variable that should be destroyed (\$-notation).	No.

Description

This function destroys variables that have previously been created using the "def" (page 1020) function. The names of the variables that should be destroyed can be provided as parameters. If no parameters are specified, the function will automatically get rid of all variables that were previously defined within the current/same namespace.

Examples

Example 1

```
{def $a=1 $b=2 $c=3}  
...  
{undef}
```

This example demonstrates how the "undef" function can be used to clean up / get rid of a previously the variables that were previously created using the "def" function .

Example 2

```
{def $a=1 $b=2 $c=3}  
...  
{undef $b}
```

This example demonstrates how the "undef" function can be used to clean up / get rid of a previously created variable. The variables \$a and \$c will not be destroyed.

5.8.4 Visualization

attribute_edit_gui (page 1032)

Outputs the edit template for a content object attribute.

attribute_pdf_gui (page 1033)

Outputs the PDF template for a content object attribute.

attribute_result_gui (page 1034)

Outputs the result template for a content object attribute.

attribute_view_gui (page 1035)

Outputs the view template for a content object attribute.

class_attribute_edit_gui (page 1036)

Outputs the edit template for a content class attribute.

class_attribute_view_gui (page 1037)

Outputs the view template for a content class attribute.

collaboration_icon (page 1038)

Outputs the icon for a collaboration item.

collaboration_participation_view (page 1039)

Outputs information about a collaboration participant.

collaboration_simple_message_view (page 1040)

Outputs the view template for a collaboration message.

collaboration_view_gui (page 1041)

Outputs the template for a collaboration item.

content_pdf_gui (page 1042)

Outputs the PDF template for a content object.

content_version_view_gui (page 1043)

Outputs a view template for a content object version.

content_view_gui (page 1044)

Outputs a view template for a content object.

event_edit_gui (page 1045)

Outputs the edit template for a workflow event.

node_view_gui (page 1046)

Outputs the view template for a node.

related_view_gui (page 1047)

Not documented yet.

shop_account_view_gui (page 1048)

Outputs the view template for a specified order.

tool_bar (page 1049)

Outputs the template for a toolbar.

attribute_edit_gui

Summary

Outputs the edit template for a content object attribute.

Usage

```
{attribute_edit_gui attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
attribute	object	The target content object attribute.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the edit interface for a content object attribute. The attribute must be specified (as a "ezcontentobjectattribute" (page 717) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

attribute_pdf_gui

Summary

Outputs the PDF template for a content object attribute.

Usage

```
{attribute_pdf_gui attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
attribute	object	The target content object attribute.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the PDF view interface for a content object attribute. The target attribute must be specified (as a "ezcontentobjectattribute" (page 717) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template

attribute_result_gui**Summary**

Outputs the result template for a content object attribute.

Usage

```
{attribute_result_gui attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
attribute	object	The target content object attribute.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the information collection result interface for a content object attribute. The attribute must be specified (as a "ezcontentobjectattribute" (page 717) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

attribute_view_gui**Summary**

Outputs the view template for a content object attribute.

Usage

```
{attribute_view_gui attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
attribute	object	The target content object attribute.	Yes.
parameters	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for a content object attribute. The attribute must be specified (as a "ezcontentobjectattribute" (page 717) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

class_attribute_edit_gui

Summary

Outputs the edit template for a content class attribute.

Usage

```
{class_attribute_edit_gui class_attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
class_attribute	object	The target content class attribute.	Yes.
other parameters	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the edit interface for a content class attribute. The attribute must be specified (as a "ezcontentclassattribute" (page 708) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

class_attribute_view_gui**Summary**

Outputs the view template for a content class attribute.

Usage

```
{class_attribute_view_gui class_attribute=attribute [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
class_attribute	object	The target content class attribute.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for a content class attribute. The attribute must be specified (as a "ezcontentclassattribute" (page 708) object) using the "attribute" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

collaboration_icon

Summary

Outputs the icon for a collaboration item.

Usage

```
{collaboration_icon collaboration_item=item [ view=view [ parameter=value  
[...] ] ]}
```

Parameters

Name	Type	Description	Required
collaboration_item	object	Collaboration item object.	Yes.
view	string	The view mode to use.	No.
parameter	any	Parameters passed to the GUI template.	No.

Description

This function outputs the icon for a collaboration item. The "collaboration_item" parameter must be used to specify the target collaboration item. The "view" parameter is optional, it can be used to specify a view mode (for example "small"). All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

collaboration_participation_view

Summary

Outputs information about a collaboration participant.

Usage

```
{collaboration_participation_view collaboration_participant=link [ view=view [ parameter=value [...] ] ]}
```

Parameters

Name	Type	Description	Required
collaboration_participant	object	The target collaboration participant object.	Yes.
view	string	The view mode to use.	No.
parameters	mixed	Parameters passed to the GUI template.	No.

Description

This function shows a collaboration participant. The "collaboration_participant" parameter must be used to specify the target collaboration participant (as an "ezcollaborationItemParticipantLink" object). The optional "view" parameter can be used to specify the view mode (for example "text_linked"). All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

collaboration_simple_message_view

Summary

Outputs the view template for a collaboration message.

Usage

```
{collaboration_simple_message_view
    sequence=sequence
    is_read=status
    item_link=message_link
    collaboration_message=simple_message
    [ view=mode ]
    [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
collaboration_message	object	An eZCollaborationSimpleMessage object.	Yes.
sequence	string	Display sequence value	Yes.
is_read	boolean	TRUE if the message has been read, FALSE if not.	Yes.
item_link	object	eZCollaborationItemMessageLink object.	Yes.
view	string	The view mode that should be used.	No.
other parameters	mixed	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the interface for a collaboration message.

collaboration_view_gui

Summary

Outputs the template for a collaboration item.

Usage

```
{collaboration_view_gui item_class=class
                        collaboration_item=item
                        [ view=mode ]
                        [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
item_class	string	The item class.	Yes.
collaboration_item	object	The collaboration item.	Yes.
view	string	The view mode that should be used.	No.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for a collaboration item. The class and the target object must be specified using the "item_class" and the "collaboration_item" parameters. The "view" parameter is optional, it can be used to specify a desired view. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

content_pdf_gui

Summary

Outputs the PDF template for a content object.

Usage

```
{content_pdf_gui content_object=object [ view=reserved ] [ parameter=value [ ... ] ]}
```

Parameters

Name	Type	Description	Required
content_object	object	The target content object.	Yes.
view	string	Reserved for future use.	No.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the PDF interface for a content object. The target object must be specified (as an "ezcontentobject" (page 712) object) using the "content_object" parameter. The "view" parameter is reserved for future use. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

content_version_view_gui

Summary

Outputs a view template for a content object version.

Usage

```
{content_version_view_gui content_version=version [ view=mode ] [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
content_version	object	The target version.	Yes.
view	string	The view mode that should be used.	No.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for a content object version. The target version must be specified (as a "ezcontentobjectversion" (page 725) object) using the "content_version" parameter. The "view" parameter is optional, it can be used to specify a desired view (for example "full", "plain", "text", etc.). The function will attempt to use the following template from within either the current design or one of the fallback designs: "templates/content/version/view/[name_of_view_mode].tpl". All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

content_view_gui

Summary

Outputs a view template for a content object.

Usage

```
{content_view_gui content_object=object [ view=view ] [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
content_object	object	The target content object.	Yes.
view	string	The view mode that should be used.	No.
other parameters	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for a content object. The target object must be specified (as an "ezcontentobject" (page 712) object) using the "content_object" parameter. The "view" parameter is optional, it can be used to specify a desired view (for example "text", "text_linked", "embed", etc.). The function will attempt to use the following template from either the current design or one of the fallback designs: "templates/content/view/[name_of_view_mode].tpl". All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

event_edit_gui

Summary

Outputs the edit template for a workflow event.

Usage

```
{event_edit_gui event=event [ parameter=value [...] ] }
```

Parameters

Name	Type	Description	Required
event	object	The target workflow event object.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the edit interface for a content object attribute. The target workflow event must be specified using the "event" parameter. All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

node_view_gui

Summary

Outputs the view template for a node.

Usage

```
{node_view_gui content_node=node [ view=view_mode [ parameter=value [ ... ] ] ]}
```

Parameters

Name	Type	Description	Required
content_node	object	The target node (as an ezcontentobjecttreenode object).	Yes.
view	string	The view mode that should be used.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function makes it possible to display a node using its view (or override) template. The target node must be specified as an "ezcontentobjecttreenode" (page 721) object using the "content_node" parameter. The "view" parameter specifies which view mode that should be used. All other parameters (of any type) will be passed on and thus become available as template variables in the view template.

Examples

Example 1

```
{def $my_node=fetch( 'content', 'node', hash( 'node_id', 96 ) )}
{node_view_gui view='example' content_node=$my_node}
```

In this example, node number 96 is fetched and stored in \$my_node. The "node_view_gui" function is used to display the target node using the "example" view mode. If there are no override rules for the specified view mode, the system will search for "example.tpl" in the "templates/node/view/" directory of the current design. If the requested template file is not found, eZ Publish will continue searching for it in the fallback designs and the standard design.

related_view_gui**Summary**

Not documented yet.

Description

Not documented, the templates seem to be missing from the distribution(s).

shop_account_view_gui

Summary

Outputs the view template for a specified order.

Usage

```
{shop_account_view_gui order=order [ view=view [ parameter=value [ ... ] ] ]}
```

Parameters

Name	Type	Description	Required
order	object	The target order object.	Yes.
view	string	The view mode that should be used.	No.
other parameters	any	Parameter(s) that will be passed to the included template.	No.

Description

This function shows the view interface for an order. The order must be specified (as a "ezorder" (page 751) object) using the "order" parameter. The "view" parameter is optional, it can be used to select a desired view mode: either "html" or "ascii". All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

tool_bar

Summary

Outputs the template for a toolbar.

Usage

```
{tool_bar name=name view=view [ parameter=value [...] ]}
```

Parameters

Name	Type	Description	Required
name	string	The name of the toolbar.	Yes.
view	string	The view mode that should be used.	Yes.
parameter	any	Parameter(s) that will be passed to the included template.	No.

Description

This function can be used to display a toolbar template. The name parameter must be used to specify the name of the toolbar that should be show. The default/standard names are "top", "right" and "bottom" - custom names are also allowed. The "view" parameter can be used to specify a desired view mode ("line", "full", etc.). All other parameters (of any type) will be passed on and thus become available as template variables in the included template.

Examples

Example 1

```
{tool_bar name='top' view='line'}
```

Shows a toolbar called "top" using the "line" view mode.

5.9 Template control structures

The template control structures are documented in the following sections:

- Conditional control (page [1051](#))
- Looping (page [1059](#))
- Deprecated (page [1057](#))

5.9.1 Conditional control

if ([page 1052](#))

Allows conditional control by the way of an IF-THEN-ELSE mechanism.

switch ([page 1054](#))

Allows conditional control of code execution.

if

Summary

Allows conditional control by the way of an IF-THEN-ELSE mechanism.

Usage

```
{if <condition>}  
...  
[ {elseif <condition>} ]  
...  
[ {else} ]  
...  
{/if}
```

Description

This construct allows for conditional execution of code fragments. It is one of the most important features of many programming languages. The eZ Publish implementation makes it possible to do conditional branching by the way of the following elements: IF, ELSE and ELSEIF. The ELSE and ELSEIF elements are optional.

Examples

Example 1

```
{if eq( $var, 128 )}  
    Hello world <br />  
{/if}
```

If \$var equals 128, the following output will be produced: "Hello world". If it does not equal 128, no output will be produced.

Example 2

```
{if eq( $var, 128 )}  
    Hello world <br />  
{else}  
    No world here, move along. <br />  
{/if}
```

If \$var equals 128, the following output will be produced: "Hello world". If it does not equal 128, the output will be "No world here, move along.".

Example 3

```
{if eq( $fruit, 'apples' )}
    Apple tree <br />
{elseif eq( $fruit, 'oranges' )}
    Orange juice <br />
{else}
    Banana split <br />
{/if}
```

If \$fruit equals "apples", the output will be "Apple tree", if it equals "oranges" then the output will be "Orange juice" - otherwise the output will be "Banana split".

switch

Summary

Allows conditional control of code execution.

Usage

```
{switch match=<variable>}

  {case match=<value>}
  {case in=<array>}
  ...
  {/case}

  {case}
  ...
  {/case}

{/switch}
```

Description

This mechanism is similar to a series of IF statements used on the same expression. This construct is typically useful when the same variable needs to be compared to different values. It executes a piece of code depending on which value that matched a given criteria. A default case should always be provided.

Please note that it is also possible to match inside arrays. This can be done by making use of the "in" argument, it is demonstrated in the last (third) example.

Examples

Example 1

```
{def $fruits='oranges'}

{switch match=$fruits}

  {case match='apples'}
    Apples <br />
  {/case}

  {case match='oranges'}
    Oranges <br />
```

```
{/case}

{case}
    Unidentified fruit! <br />
{/case}

{/switch}
```

The following output will be produced: "Oranges".

Example 2

```
{def $fruits='Hello world'}

{switch match=$fruits}

{case match='apples'}
    Apples <br />
{/case}

{case match='oranges'}
    Oranges <br />
{/case}

{case}
    Unidentified fruit! <br />
{/case}

{/switch}
```

The following output will be produced: "Unidentified fruit!" - which is the outcome of the default case (none of the other cases matched).

Example 3

```
{def $digit=1}

{switch match=$digit}

{case in=array( 1, 2 )}
    This one matches.
{/case}
```

```
{case in=array( 2, 3 )}  
    This one does not match.  
{/case}  
  
{case}  
    Not this one either.  
{/case}  
  
{/switch}
```

The following output will be produced: "This one matches.".

5.9.2 Deprecated

section (page 1058)

Deprecated looping, branching, etc.

section**Summary**

Deprecated looping, branching, etc.

Description

Refer to the [documentation of the old template syntax](#) for more information about this control structure.

Please note that this control structure has been deprecated and should not be used. It is included here only because some of the default templates in the distributions are still using it.

5.9.3 Looping

do (page 1060)

Creates a do...while loop.

for (page 1061)

Creates a generic for loop.

foreach (page 1062)

Iterates over arrays in different ways.

while (page 1064)

Creates a while loop.

do**Summary**

Creates a do...while loop.

Usage

```
{do}
  [ {delimiter}...{/delimiter} ]
  [ {break}      ]
  [ {continue}   ]
  [ {skip}       ]
{/do while <condition> [ sequence <array> as $seqVar ]}
```

Description

This mechanism is very similar to the "while" (page 1062) construct, except that the expression is checked at the end of each iteration instead of in the beginning. The main difference is that this construct will always execute the first iteration (regardless of how the test expression evaluates). It supports breaking, continual and skipping.

Examples**Example 1**

```
{do}
  Keep printing this line ($counter) <br />
  {set $counter=inc( $counter )}

{/do while ne( $counter, 8 )}
```

If the initial value of \$counter is 0, the following output will be produced:

```
Keep printing this line (0)
Keep printing this line (1)
Keep printing this line (2)
Keep printing this line (3)
Keep printing this line (4)
Keep printing this line (5)
Keep printing this line (6)
Keep printing this line (7)
Keep printing this line (8)
```

for**Summary**

Creates a generic for loop.

Usage

```
{for <number> to <number> as $itemVar [ sequence <array> as $seqVar ]}
  [ {delimiter}...{/delimiter}]
  [ {break}    ]
  [ {continue} ]
  [ {skip}     ]
{/for}
```

Description

This mechanism makes it possible to do generic looping. It supports looping over numerical ranges in both directions. In addition it also supports breaking, continual and skipping.

Examples**Example 1**

```
{for 0 to 7 as $counter}

  Value of counter: {$counter} <br />

{/for}
```

The following output will be produced:

```
Value of counter: 0
Value of counter: 1
Value of counter: 2
Value of counter: 3
Value of counter: 4
Value of counter: 5
Value of counter: 6
Value of counter: 7
```

foreach

Summary

Iterates over arrays in different ways.

Usage

```
{foreach <array> as [ $keyVar => ] $itemVar
    [ sequence <array> as $sequenceVar ]
    [ offset <offset> ]
    [ max <max> ]
    [ reverse ]
    [ {delimiter}...{/delimiter} ]
    [ {break} ]
    [ {continue} ]
    [ {skip} ]
}

{/foreach}
```

Description

This construct makes it possible to iterate over arrays in different ways. The loop can be tweaked using the parameters (see above).

Examples

Example 1

```
{foreach $objects as $object}
    {$object.name} <br />
{/foreach}
```

This example will print out the names of the objects that are stored in the \$objects array. If this array stores 4 objects with the following names: "Emmett Brown", "Marty McFly", "Lorraine Baines" and "Biff Tannen", the following output will be produced:

```
Emmett Brown
Marty McFly
Lorraine Baines
Biff Tannen
```

Example 2

```
{foreach $objects as $index => $object}  
  
    {$index} : {$object.name} <br />  
  
{/foreach}
```

This example demonstrates how to create an iteration counter.

```
0: Emmett Brown  
1: Marty McFly  
2: Lorraine Baines  
3: Biff Tannen
```

Example 3

```
{foreach $objects as $object sequence array( 'dark', 'light' ) as $style}  
  
    <div class="{$style}">{$object.name}</div>  
  
{/foreach}
```

This example demonstrates how to create a loop where the iterations are displayed using alternating styles (in this case dark, light, dark, light and so on).

while

Summary

Creates a while loop.

Usage

```
{while <condition> [ sequence <array> as $seqVar ]}  
  [ {delimiter}...{/delimiter} ]  
  [ {break} ]  
  [ {continue} ]  
  [ {skip} ]  
{/while}
```

Description

This construct is the simplest loop mechanism that the template language offers. It tells eZ Publish to execute the nested statement(s) repeatedly, as long as a given expression evaluates to TRUE. The value of the expression is checked for every loop iteration (at the beginning of the iteration). If the given expression evaluates to FALSE from the very beginning, the nested statement(s) will not be executed.

Examples

Example 1

```
{while ne( $counter, 8 )}  
  
  Print this line eight times ($counter) <br />  
  {set $counter=inc( $counter )}  
  
{/while}
```

If the initial value of \$counter is zero, the following output will be produced:

```
Print this line eight times (0)  
Print this line eight times (1)  
Print this line eight times (2)  
Print this line eight times (3)  
Print this line eight times (4)  
Print this line eight times (5)  
Print this line eight times (6)  
Print this line eight times (7)
```

5.10 Template override conditions

This section contains an overview of the override conditions that can be used to override the system templates.

Override example

```
[magic_pockets]          (1)
Source=node/view/full.tpl (2)
MatchFile=magic_pocket.tpl (3)
Subdir=templates          (4)
Match[class_identifier]=pocket (5)
Match[section]=34          (6)
...
...
```

1. The name of the override.
2. The template that should be overridden.
3. The alternate template that should be used.
4. The location of the override templates.
5. Match condition #1
6. Match condition #2

The following sections reveal the override rules for the different template files:

- class/edit.tpl (page [1067](#))
- class/groupedit.tpl (page [1068](#))
- class/view.tpl (page [1069](#))
- content/advancedsearch.tpl (page [1070](#))
- content/browse.tpl (page [1071](#))
- content/collectedinfo/*.tpl (page [1072](#))
- content/collectedinfo/*.tpl (page [1073](#))
- content/collectedinfomail/*.tpl (page [1074](#))
- content/datatype/edit/*.tpl (page [1075](#))
- content/datatype/view/*.tpl (page [1076](#))

- content/edit.tpl (page [1077](#))
- content/search.tpl (page [1078](#))
- content/versions.tpl (page [1079](#))
- content/versionview.tpl (page [1080](#))
- layout/set.tpl (page [1081](#))
- node/view/*.tpl (page [1082](#))
- node/view/pdf.tpl (page [1083](#))
- pagelayout.tpl (page [1084](#))
- workflow/edit.tpl (page [1085](#))
- workflow/groupedit.tpl (page [1086](#))
- workflow/view.tpl (page [1087](#))

5.10.1 class/edit.tpl

Module/view: edit

Condition	Description
class	Matches the ID number of the class.

5.10.2 class/groupedit.tpl

Module/view: groupedit

Condition	Description
object	Matches the ID number of class group.

5.10.3 class/view.tpl

Module/view: view

Condition	Description
class	Matches the ID number of the class.
class_identifier	Matches the identifier of the class (for example "folder").

5.10.4 content/advancedsearch.tpl

Module/view: advancedsearch

Condition	Description
section	Matches the number of the section in which the search was conducted.

5.10.5 content/browse.tpl

Module/view: browse

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
view_offset	Matches the offset view parameter.
depth	Matches the depth of the node. The depth of a top level node is 1.
navigation_part_identifier	Matches the identifier of the navigation part.
url_alias	Matches the virtual URL of the node.

5.10.6 content/collectedinfo/*.tpl

Module/view: collectinformation

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
depth	Matches the depth of the node. The depth of a top level node is 1.
navigation_part_identifier	Matches the identifier of the navigation part.
url alias	Matches the virtual URL of the node.

5.10.7 content/collectedinfo/*.tpl

Module/view: collectedinfo

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
depth	Matches the depth of the node. The depth of a top level node is 1.
navigation_part_identifier	Matches the identifier of the navigation part.
url_alias	Matches the virtual URL of the node.

5.10.8 content/collectedinfomail/*.tpl

Module/view: collectinformation

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
depth	Matches the depth of the node. The depth of a top level node is 1.
navigation_part_identifier	Matches the identifier of the navigation part.
url alias	Matches the virtual URL of the node.

5.10.9 content/datatype/edit/*.tpl

Module/view: none.

Condition	Description
class_identifier	Matches the identifier of the class.
attribute_identifier	Matches the identifier of the attribute.

5.10.10 content/datatype/view/*.tpl

Module/view: none.

Condition	Description
class_identifier	Matches the identifier of the class.
attribute_identifier	Matches the identifier of the attribute.

5.10.11 content/edit.tpl

Module/view: edit

Condition	Description
object	Matches the ID number of the object.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
class_group	Matches the ID number of the group that the class which the object is an instance of belongs to.

5.10.12 content/search.tpl

Module/view: search

Condition	Description
section	Matches the number of the section in which the search was conducted.

5.10.13 content/versions.tpl

Module/view: versions

Condition	Description
object	Matches the ID number of the object.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
section_id	Matches the number of the section which the object belongs to.

5.10.14 content/versionview.tpl

Module/view: versionview

Condition	Description
navigation_part_identifier	Matches the identifier of the navigation part.

5.10.15 layout/set.tpl

Module/view: set

Condition	Description
layout	Matches the name of the layout (for example "print", "fullpage", etc.).

5.10.16 node/view/*.tpl

Module/view: view

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
view_offset	Matches the offset view parameter.
depth	Matches the depth of the node. The depth of a top level node is 1.
section	Matches the number of the section which the object belongs to.
navigation_part_identifier	Matches the identifier of the navigation part.
viewmode	Matches the name of the view mode (full, line, etc.)
url_alias	Matches the virtual URL of the node.
persistent_variable	Matches the value of the persistent variable.
class_group	Matches the ID number of the group that the class which the object is an instance of belongs to.

5.10.17 node/view/pdf.tpl

Module/view: pdf

Condition	Description
object	Matches the ID number of the object.
node	Matches the ID number of the node.
parent_node	Matches the ID number of the parent node.
class	Matches the ID number of the class which the object is an instance of.
class_identifier	Matches the identifier of the class which the object is an instance of.
depth	Matches the depth of the node. The depth of a top level node is 1.
url_alias	Matches the virtual URL of the node.
class_group	Matches the ID number of the group that the class which the object is an instance of belongs to.

5.10.18 pagelayout.tpl

Module/view: none.

Condition	Description
[any]	The pagelayout may be overridden using the same keys that are available for the view which was used within the request. For example, if the "view" view of the "content" module was accessed, then the override conditions of that view will be matched. This makes it possible to create a pagelayout override that will be triggered when for example nodes referencing objects that belong to a certain section are requested.

5.10.19 workflow/edit.tpl

Module/view: edit

Condition	Description
workflow	Matches the ID number of the workflow.

5.10.20 workflow/groupedit.tpl

Module/view: groupedit

Condition	Description
workflowgroup	Matches the ID number of the workflow group.

5.10.21 workflow/view.tpl

Module/view: view

Condition	Description
workflow	Matches the ID number of the workflow.

5.11 Template fetch functions

The template fetch functions are documented in the following sections:

- class (page [373](#))
- collaboration (page [392](#))
- content (page [407](#))
- error (page [527](#))
- ezinfo (page [528](#))
- form (page [533](#))
- infocollector (page [536](#))
- layout (page [541](#))
- notification (page [546](#))
- package (page [558](#))
- pdf (page [581](#))
- reference (page [585](#))
- role (page [586](#))
- rss (page [594](#))
- search (page [600](#))
- section (page [603](#))
- setup (page [617](#))
- shop (page [618](#))
- trigger (page [646](#))
- url (page [649](#))
- user (page [658](#))
- workflow (page [683](#))

5.12 Template PDF functions

anchor (page 1091)

Creates an internal link anchor.

create_index (page 1092)

Creates an index based on the specified keywords.

filled_circle (page 1093)

Creates a filled circle.

filled_rectangle (page 1095)

Creates a filled rectangle.

footer (page 1097)

Sets the footer.

footer_block (page 1099)

Inserts complex footers.

frame_header (page 1100)

Sets the header text.

frontpage (page 1102)

Adds content to the frontpage.

header (page 1103)

Inserts a header / title.

header_block (page 1105)

Inserts a complex header.

image (page 1106)

Inserts an image to the PDF document.

keyword (page 1108)

Adds a keyword to the keyword index.

line (page 1109)

Draws a line.

link (page 1111)

Inserts an external link.

new_line (page 1112)

Inserts a new line.

new_page (page 1113)

Inserts a new page.

page_number (page 1114)

Starts the page number counter.

set_font (page 1115)

Changes the default font.

set_margin (page 1117)

Sets the page margins or line spacing.

strike (page 1119)

Inserts striked text.

table (page 1120)

Inserts a table.

text (page 1122)

This function inserts formatted text into the PDF document.

text_box (page 1125)

Inserts text at a specified location.

text_frame (page 1127)

Inserts a text with a frame.

toc (page 1129)

Inserts a generated table of contents.

ul (page 1130)

Inserts a bullet list.

5.12.1 anchor

Summary

Creates an internal link anchor.

Usage

```
{pdf( anchor, label )}
```

Parameters

Name	Type	Description	Required
label	string	The name of the anchor.	Yes.

Description

This function does not work. It is supposed to create an anchor for making internal links in PDF documents.

Examples

Example 1

```
{pdf( 'anchor', 'image_1' )}
```

5.12.2 create_index

Summary

Creates an index based on the specified keywords.

Usage

```
pdf( create_index )
```

Description

This function creates an index page from a collection of keywords. The keywords must be specified using the "keyword" (page 1108) PDF template function. The index will start on the next page, and consists of the specified keywords with their corresponding page number.

Examples

Example 1

```
{pdf( 'text', 'Index page demo.' | wash( 'pdf' ) )}
{pdf( 'keyword', 'apples' )}
{pdf( 'keyword', 'bananas' )}
{pdf( 'create_index' )}
```

This example creates an index page using the following keywords "apples" and "bananas".

5.12.3 filled_circle

Summary

Creates a filled circle.

Usage

```
pdf( filled_circle, hash( radius, circle_radius,
                           x,      x_position,
                           y,      y_position,
                           [ rgb,   rgb_color, ]
                           [ cmyk, cmyk_color ] ) )
```

Parameters

Name	Type	Description	Required
radius	integer	The radius of the circle.	Yes.
x	integer	The x position of the circle placement.	Yes.
y	integer	The y position of the circle placement.	Yes.
rgb	array	The RGB color of the circle.	Only if the "cmyk" parameter is omitted.
cmyk	array	The CMYK color of the circle.	Only if the "rgb" parameter is omitted.

Description

This function creates a filled circle with a given radius and color. The position of the circle is specified with the "x" and "y" parameters. The given coordinate must be within the page margins. The lower-left corner is at (0,0).

Examples

Example 1

```
{pdf( 'new_page' )}

{pdf( 'filled_circle', hash( 'radius', 100,
                           'x',      300,
                           'y',      400,
                           'rgb',    array( 0, 0, 0 ) ) }

{pdf( 'filled_circle', hash( 'radius', 98,
                           'x',      300,
                           'y',      400,
                           'rgb',    array( 255, 0, 0 ) ) }
```

This example creates a black circle with red filling on a new page.

5.12.4 filled_rectangle

Summary

Creates a filled rectangle.

Usage

```
pdf( filled_rectangle, hash( x,           x_position,
                           y,           y_position,
                           width,       width,
                           height,      height,
                           [ rgb,        rgb_color,      ]
                           [ rgbTop,     rgb_top_color,
                             rgbBottom,   rgb_bottom_color, ]
                           [ cmyk,       cmyk_color,      ]
                           [ cmykTop,    cmyk_top_color,
                             cmykBottom,  cmyk_bottom_color, ] ) )
```

Parameters

Name	Type	Description	Required
x	integer/ float	X coordinate.	Yes.
y	integer/ float	Y coordinate.	Yes.
width	integer/ float	Rectangle width.	Yes.
height	integer/ float	Rectangle height.	Yes.
rgb	array	Rectangle RGB color.	No.
rgbTop	array	Rectangle top RGB color.	No.
rgbBottom	array	Rectangle bottom RGB color.	No.
cmyk	array	Rectangle CMYK color.	No.
cmykTop	array	Rectangle top CMYK color.	No.
cmykBottom	array	Rectangle bottom CMYK color.	No.

Description

This function creates a filled rectangle. The "x" and "y" parameters specify the lower-left corner of the rectangle. The "width" and "height" parameters determine the size of the rectangle.

The color of the rectangle can be specified using either the "rgb" or the "cmyk" parameter. The "rgb" parameter must be an array consisting of three numbers between 0 and 255. The "cmyk" parameter must be an array of four floats between 0 and 1. The "rgb" or "cmyk" parameter

fills the entire rectangle with the same color; no color interpolation. This option has the same result as setting the "rgbTop", "rgbBottom" or "cmykTop", "cmykBottom" with the same color. A different top and bottom color results in a gradient fill. The color will be linearly interpolated from the top color to the bottom color of the rectangle.

This function uses features that appeared in the PDF1.3 specification. Because of this, rectangles will not visible in PDF-viewers that are not PDF 1.3 compliant (for example "Ghostview", etc.).

Examples

Example 1

```
{pdf( 'filled_rectangle', hash( 'x', -10,
                                'y', 30.5,
                                'width', 282.9,
                                'height', 821.9,
                                'cmykTop', array( 0.96, 0.30, 0, 0 ),
                                'cmykBottom', array( 0.15, 0.04, 0, 0 ) )}

{pdf( 'filled_rectangle', hash( 'x', 272.9,
                                'y', 30.5,
                                'width', 333,
                                'height', 821.9,
                                'cmykTop', array( 0.34, 0.05, 0, 0 ),
                                'cmykBottom', array( 0.10, 0.02, 0, 0 ) ) }

{pdf( 'filled_rectangle', hash( 'x', -10,
                                'y', -10,
                                'width', 606,
                                'height', 40.5,
                                'cmyk', array( 0.89, 0.43, 0.01, 0 ) ) }
```

This example fills the page with three different rectangles.

5.12.5 footer

Summary

Sets the footer.

Usage

```
pdf(footer, hash( [ text,           text,
                    [ align,        text_alignment,
                      ] ]
                  [ font,         font_name,
                    ] ]
                  [ size,         font_size,
                    ] ]
                  [ page,          page_interval,
                    ] ]
                  [ pageOffset,   page_offset,
                    ] ]

                  [ margin,       hash( [ bottom, bottom_margin, ]
                                [ left,    left_margin,   ]
                                [ right,   right_margin, ] ), ]
                    ] )

                  [ line,         hash( [ leftMargin,  left_margin,   ]
                                [ rightMargin, right_margin, ]
                                [ thickness,   line_thickness, ] ), ]
                    ] )

                  [ newline, boolean
                    ] ) )
```

Parameters

Name	Type	Description	Required
text	string	Text used in the footer.	No.
align	string	Alignment of the text.	No.
page	string	Page occurrence is all, even or odd.	No.
pageOffset	integer	Page number where the footer starts.	No.
size	integer	Size of the font.	No.
font	string	Type of the font.	No.
margin	hash	The definition of the margins.	No.
left	integer/ float	Left footer margin.	No.
right	integer/ float	Right footer margin.	No.
bottom	integer/ float	Bottom footer margin.	No.
line	hash	The definition of a line.	No.
leftMargin	integer/ float	Left line margin.	No.
rightMargin	integer/ float	Right line margin.	No.
thickness	integer/ float	The thickness of the line.	No.
newline	boolean	Adds a new line to the footer. Force footer entry to new line	No.

Description

This function makes it possible to create a footer for a collection of pages. Please note that it does not work correctly. The page number from where the footer starts can be specified using the "pageOffset" parameter. The "page" parameters can be used to specify whether the footer should occur on "all", "even" or "odd" pages.

If the "text" parameter is used, the parameters "align", "type", and "size" can be used to change the default alignment, font type and size. Please refer to the "text" (page 1122) function for more information about these parameters.

There are two keywords reserved in the text string (parameter):

- #page: will be replaced by the current page number.
- #total: will be replaced by the total number of pages.

The footer should only be included once in the PDF document.

5.12.6 footer_block

Summary

Inserts complex footers.

Usage

```
{pdf( footer_block, hash( block_code, $variable ) )}
```

Parameters

Name	Type	Description	Required
block_code	mixed	Variable of template code set using "set_block".	Yes.

Description

This function makes it possible to create complex footers. There are some known problems with this function.

5.12.7 frame_header

Summary

Sets the header text.

Usage

```
{pdf(frame_header, hash( [ text, text, ]
    [ align, text_alignment, ]
    [ page, page_interval, ]
    [ pageOffset, page_offset, ]
    [ size, font_size, ]
    [ font, font_name, ]
    [ margin, hash( [ bottom, bottom_margin,
        [ left, left_margin, ]
        [ right, right_margin, ] ), ]
    [ line, hash( [ leftMargin, left_margin,
        [ rightMargin, right_margin, ]
        [ thickness, line_thickness, ] ), ] ) ) }
```

Parameters

Name	Type	Description	Required
text	string	The header text.	No.
align	string	The text alignment.	No.
page	string	Page occurrence ("all", "even" or "odd").	No.
pageOffset	integer	Page number to start footer at.	No.
size	integer	The font size.	No.
font	string	The font name.	No.
margin	hash	Margin definition.	No.
left	integer/ float	Left header margin.	No.
right	integer/ float	Right header margin.	No.
bottom	integer/ float	Bottom footer margin.	No.
line	hash	Line definition.	No.
leftMargin	integer/ float	Left line margin.	No.
rightMargin	integer/ float	Right line margin.	No.
thinkness	integer/ float	Line thickness.	No.
newline	boolean	Force header entry to new line.	No.

Description

This function is deprecated and should not be used.

5.12.8 frontpage

Summary

Adds content to the frontpage.

Usage

```
pdf( frontpage, hash( text,
                      [ align,           text_alignment,   ]
                      [ size,            text_size,        ]
                      [ top_margin,      top_text_margin ] ) )
```

Parameters

Name	Type	Description	Required
text	string	Text that will be added to the front page.	Yes.
align	string	The text alignment.	No.
size	int	The font size.	No.
top_margin	int	The top margin.	No.

Description

This function makes it possible to add text to the front page. If the frontpage is not present, a new front page will be created when the function is used for the first time. This function should be called after all other content has been added to the PDF document. The "align" and "size" parameters control the alignment and the size of the text. The "top_margin" parameter can be used to specify the top margin (vertical positioning) for the text.

Examples

Example 1

```
{pdf( 'frontpage', hash( 'text', 'eZ Publish
                           | wash( 'pdf' ), 'align', 'center', 'size', 32, 'top_margin', 350 ) )

{pdf( 'frontpage', hash( 'text', 'The road ahead' | wash( 'pdf' ), 'align',
                           'center', 'size', 22, 'top_margin', 400 ) )}
```

This example adds a main and a sub title to the front page. The sub title is smaller and positioned below the main title. Both titles are centered. Notice that the main title ends with a new line, this assures that the sub title is correctly centered.

5.12.9 header

Summary

Inserts a header / title.

Usage

```
pdf(header, hash( text, header_text,
                  level, header_level,
                  size, font_size,
                  [ align, text_alignment, ]
                  [ font, font_type ] ) )
```

Parameters

Name	Type	Description	Required
text	string	The header text.	Yes.
level	integer	The header level.	Yes.
size	integer	The font size.	Yes.
align	string	The text alignment.	No.
font	string	The font that should be used.	No.

Description

This function inserts a header or title. The main difference between the "text" and the header function is the "level" parameter. This parameter specifies the type of the header. The level number goes from 1 and up:

1. Chapters
2. Section
3. Subsection
4. Subsubsection
5. etc.

The level has nothing to do with the header text, it controls how the text appears in the table of contents (page [1129](#)). The font size, font type, and text alignment can be specified using the "size", "font", and "align" parameters. Please refer to the documentation of the "text" function for an explanation of these parameters.

Examples

Example 1

```
{pdf( 'header', hash( 'level', 1, 'text', 'The first chapter'|wash( 'pdf' ),  
'size', 20 ) )}  
{pdf( 'header', hash( 'level', 2, 'text', 'The first section'|wash( 'pdf' ),  
'size', 16 ) )}  
{pdf( 'header', hash( 'level', 2, 'text', 'The second section'|wash( 'pdf' ),  
'size', 16 ) )}  
{pdf( 'header', hash( 'level', 3, 'text', 'The first subsection'|wash( 'pdf' ),  
'size', 14 ) )}  
{pdf( 'header', hash( 'level', 1, 'text', 'The second chapter'|wash( 'pdf' ),  
'size', 20 ) )}
```

This example shows how to create a chapters, section and subsections.

5.12.10 header_block

Summary

Inserts a complex header.

Usage

```
{pdf( header_block, hash( block_code, $variable ) )}
```

Parameters

Name	Type	Description	Required
block_code	mixed	Variable of template code set using the "set_block" function.	Yes.

Description

This function is not documented because there are some problems with it.

5.12.11 image

Summary

Inserts an image to the PDF document.

Usage

```
pdf( image, hash( src,      image_path,
                  [ width,  image_width,      ]
                  [ height, image_height,     ]
                  [ align,   justification,   ]
                  [ x,       x_position,     ]
                  [ y,       y_position,     ]
                  [ dpi,    resolution,     ]
                  [ static, flow_properties ] ) )
```

Parameters

Name	Type	Description	Required
src	string	The path to the image.	Yes.
width	integer	The width of the image.	No.
height	integer	The height of the image.	No.
align	string	The alignment of the image.	No.
x	integer	The absolute horizontal offset.	No.
y	integer	The absolute y offset.	No.
dpi	integer	The image resolution.	No.
static	boolean	Float properties of the image.	No.

Description

This function makes it possible to insert an image. The image location or image source points to the root directory of the eZ publish installation. The image location is relative to this directory. Currently, the supported image types are "jpg" and "png" without alpha channel.

The width and height of the image can be specified using the "width" and "height" parameters. If these parameters are omitted, the image will be scaled to 100 by 100 dots. The "align" parameter can be used to specify the horizontal position of the image. Possible alignments are "left", "right", and "center" - the default is "left". The "x" and "y" parameter makes it possible to place the image at an exact location. These parameters will override the settings from the image alignment ("align") and float properties ("static").

If the image is too sharp, the dots-per-inch can be decreased using the "dpi" parameter. (It is not possible to make the image sharper than the original.)

The "static" parameter can be used to specify whether the content around the image will float

around the image or not. When the "static" parameter is set to TRUE, additional content will not float around and may actually overlap the image.

Examples

Example 1

```
{pdf( 'text', 'The logo is on the right hand side of this text.' |wash( 'pdf' ) )
}{pdf( 'image', hash( 'src', 'design/mysite/images/company_logo.png',
                     'width', 200,
                     'height', 200,
                     'align', 'right' ) )}
```

This example displays an image with some text on the left hand side.

Example 2

```
{pdf( 'image', hash( 'src', $image.full_path, 'width', $image.width, 'height',
                     $image.height ) )}
```

This example adds the image assigned to the \$image variable.

5.12.12 keyword

Summary

Adds a keyword to the keyword index.

Usage

```
pdf( keyword, word )
```

Parameters

Name	Type	Description	Required
word	string	The keyword that should be added.	Yes.

Description

This function adds a word to the keywords. When generating the keyword index (using the "create_index" (page 1092) function), an internal link to the position where the keyword was added will be created.

Examples

Example 1

```
{pdf( 'keyword', 'alien' )}
```

This example will add the "alien" word to the keywords.

5.12.13 line

Summary

Draws a line.

Usage

```
pdf( line, hash( x1,
                  x_start,
                  y1,          y_start,
                  x2,          x_stop,
                  y2,          y_stop,
                  [ page,       occurrence,      ]
                  [ thickness, line_thickness ] ) )
```

Parameters

Name	Type	Description	Required
x1	float	Start coordinate of the X-ax.	Yes.
y1	float	Start coordinate of the Y-ax.	Yes.
x2	float	Stop coordinate of the X-ax.	Yes.
y2	float	Stop coordinate of the Y-ax.	Yes.
pages	string	The pages on which the line should appear.	No.
thickness	float	The thickness of the line.	No.

Description

This function draws a line. The line will be drawn from the (x1, y1) coordinate to the (x2, y2) coordinate. If the "pages" parameter is omitted or set to "current", the line will only be present on the current page. If "pages" is set to all, it will appear on all the pages. The thickness of the line can be set with the "thickness" parameter; the default thickness is 1.

When the line must be visible on all the pages, the line definition must be set after the content is written to the PDF-document (at the end of the template) and the line should be defined only once. A common technique is to use the "include" (page 1012) operator.

Examples

Example 1

```
{pdf( 'line', hash( 'x1', 100, 'y1', 100, 'x2', 100, 'y2', 300 ) )}
```

This example draws a vertical line, from (100, 100) to (100, 300) on the current page.

Example 2

```
{pdf( 'line', hash( 'x1', 20, 'y1', 30, 'x2', 20, 'y2', 40, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 20, 'y1', 40, 'x2', 30, 'y2', 40, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 30, 'y1', 40, 'x2', 30, 'y2', 35, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 25, 'y1', 35, 'x2', 33, 'y2', 35, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 33, 'y1', 35, 'x2', 33, 'y2', 27, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 25, 'y1', 35, 'x2', 25, 'y2', 27, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 25, 'y1', 27, 'x2', 33, 'y2', 27, pages, 'all' ) )}  
{pdf( 'line', hash( 'x1', 20, 'y1', 30, 'x2', 25, 'y2', 30, pages, 'all' ) )}
```

This example draws the eZ logo in the lower left corner of every page.

5.12.14 link

Summary

Inserts an external link.

Usage

```
pdf( link, hash( url, url,
                  text, link_text ) )
```

Parameters

Name	Type	Description	Required
url	string	The URL that should be called.	Yes.
text	string	The text of the link.	Yes.

Description

This function insert a link to an external document.

Examples

Example 1

```
{pdf( 'link', hash( 'url', 'http://www.ez.no',
                     'text', 'eZ Systems website' ) )}
```

This example creates an link to the eZ Systems website.

5.12.15 new_line

Summary

Inserts a new line.

Usage

```
pdf( new_line )
```

Description

This function inserts a new line.

Examples

Example 1

```
{pdf( 'text', 'Some text.'|wash( 'pdf' ) )}
{pdf( 'text', 'This sentence is written at the same line.'|wash( 'pdf' ) )}
{pdf( 'new_line' )}
{pdf( 'text', 'The start of a new line.'|wash( 'pdf' ) )}
{pdf( 'new_line' )}
{pdf( 'new_line' )}
{pdf( 'text', 'There is a blank line above this one.'|wash( 'pdf' ) )}
```

This example demonstrates everything that can be done with the "new_line" function.

5.12.16 new_page

Summary

Inserts a new page.

Usage

```
pdf( new_page )
```

Description

This function inserts a new page, following content will be placed on a new page. Please note that the system automatically takes care of adding new pages when needed. This function simply makes it possible to manually insert a new page.

Examples

Example 1

```
{pdf( 'text', 'Text that will be on page #1.' )}
{pdf( 'new_page' )}
{pdf( 'header', hash( 'text', 'Text that will be on page #2.', 'level', 1,
'size', 18 ) )}
```

This example demonstrates the manual insertion of a new page.

5.12.17 page_number

Summary

Starts the page number counter.

Usage

```
{pdf( page_number )}
```

Description

This function starts the page number counter.

5.12.18 set_font

Summary

Changes the default font.

Usage

```
pdf(set_font, hash( [ name,           font_name, ]
                     [ size,            font_size, ]
                     [ colorCMYK,      cmyk_color,]
                     [ colorRGB,        rgb_color, ]
                     [ justification,   text_justification ] ) )
```

Parameters

Name	Type	Description	Required
name	string	The font type (name).	No.
size	integer	The size of the font.	No.
colorCMYK	array	The CMYK color.	No.
colorRGB	array	The RGB color.	No.
justification	string	The text alignment.	No.

Description

This function makes it possible to change the default/current font and the text color, style, etc. The following list shows the fonts that can be used.

- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- Courier
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Helvetic
- Symbol
- Times-Bold

- Times-BoldItalic
- Times-Italic
- Times-Roman

Either the "colorRGB" or the "colorCMYK" parameter can be used to specify the font color. The "colorRGB" parameter must be an array consisting of three integers between 0 and 255. The "colorCMYK" parameter must be an array of four decimal values between 0 and 1.

The justification parameter controls the text alignment, it can be:

- left
- right
- center
- full

Examples

Example 1

```
{pdf( 'set_font', hash( 'name', 'Times-Roman', 'size', 12 ) )}

{pdf( 'text', 'Hello world'|wash( 'pdf' ) )}

{pdf( 'new_line' )}
{pdf( 'new_line' )}
{pdf( 'set_font', hash( 'name', 'Times-Italic',
                      'size', 10,
                      'justification', 'full',
                      'colorRGB', array( 255, 255, 128 ) ) }

{pdf( 'text', 'Example'|wash( 'pdf' ) )}
```

This example demonstrates how to change the default font type, size, style, and color.

5.12.19 set_margin

Summary

Sets the page margins or line spacing.

Usage

```
pdf( set_margin, hash( [ left,           left_margin,   ]
                      [ right,          right_margin,  ]
                      [ top,            top_margin,   ]
                      [ bottom,         bottom_margin, ]
                      [ x,              x_offset,     ]
                      [ y,              y_offset,     ]
                      [ line_space,    line_space ] ) )
```

Parameters

Name	Type	Description	Required
left	float	Left page margin.	No.
right	float	Right page margin.	No.
top	float	Top page margin.	No.
bottom	float	Bottom page margin.	No.
x	float	Page x offset.	No.
y	float	Page y offset.	No.
line_space	float	Line space size.	No.

Description

This function sets the page margins of the current and following pages. The "left", "right", "top", and "bottom" parameters specify the distances from the edges. The parameters "x" and "y" specify a point on the current page where the new margin starts. For the next pages, the parameters "x" and "y" are ignored and thus the margins will affect the entire area of the upcoming pages. The "line_space" parameter specifies the amount of white space between each line. The default margins are configured in the "pdf.ini" file.

Examples

Example 1

```
{pdf( 'set_margin', hash( 'left', 370, 'right', 100 ) )}
{pdf( 'text', 'The text written on this page is presented in a small column.
The space between each new line is also adjusted.' |wash( 'pdf' ) )}
```

This example creates a small column with some text.

5.12.20 strike

Summary

Inserts striked text.

Usage

```
pdf( strike, text )
```

Parameters

Name	Type	Description	Required
text	string	Text to strike through.	Yes.

Description

This function draws a line through the specified text. The "set_font" (page 1115) function is the only way to change the font type, font size, font color, and text alignment.

Examples

Example 1

```
{pdf( 'strike', 'A strike is a deliberate absence from work.' | wash( 'pdf' ) )}
```

This example demonstrates a striked sentence.

5.12.21 table

Summary

Inserts a table.

Usage

```
pdf(table, table_rows, hash( [ showLines,           number,           ]
                           [ firstRowTitle,      boolean,          ]
                           [ titleCellCMYK,      cmyk_color,       ]
                           [ titleTextCMYK,      cmyk_color,       ]
                           [ titleFontSize,      font_size,        ]
                           [ cellCMYK,           cmyk_color,       ]
                           [ textCMYK,            cmyk_color,       ]
                           [ rowGap,              row_gap,          ]
                           [ colGap,              col_gap,          ]
                           [ cellPadding,         cell_padding,     ]
                           [ width,                table_width,     ]
                           [ repeatTableHeader,   repeat_table_header ] ) )
```

Parameters

Name	Type	Description	Required
showLines	integer	Defines the drawing style of the table.	No.
firstRowTitle	boolean	Defines whether the first row should be displayed as a title header (or not).	No.
titleCellCMYK	array	CMYK color of the title cell.	No.
titleTextCMYK	array	CMYK color of the title text.	No.
titleFontSize	integer	Font size of the title.	No.
cellCMYK	array	CMYK color of the cells.	No.
textCMYK	array	CMYK color of the text.	No.
rowGap	integer/ float	Space between the table rows.	No.
colGap	integer/ float	Space between the table columns.	No.
cellPadding	integer/ float	Padding around the text in each cell.	No.
width	integer/ float	Width of the table.	No.
repeatTableHeader	boolean	Defines whether the table header should be repeated on a new page.	No.

Description

This function inserts a table. The table data must be defined using the "table_rows" parameter. Use the "set-block" (page 1027) template function to specify the rows and the cells. The rows and cells are defined the same way as rows and cells are defined in HTML. Note that nested tables are not supported.

The "showLines" parameter specifies the drawing style of the table, it must be one of the following numbers:

- 0 - Don't draw lines at all.
- 1 - Draw only the inner lines.
- 2 - Draw all the lines.

The "firstRowTitle" expects a boolean value. This parameter defines whether the first row in the table, should be used for column descriptions. With the "titleCellCMYK" and "titleTextCMYK" parameters the color of the cell-background and the text of the column titles can be controlled. The title font size can be changed with the "titleFontSize" parameter.

For the rest of the cells can the background and text color be changed with the "cellCMYK" and "textCMYK".

The size of the table and the space between the columns and rows is defined with the "rowGap", "colGap", "cellPadding", and "width" parameters.

The parameter "repeatTableHeader" defines whether the title should be repeated on each new page, if the table is stretched over multiple pages.

5.12.22 text

Summary

This function inserts formatted text into the PDF document.

Usage

```
pdf( text, page_text, hash( [ font, font_name,
                             [ size, font_size,
                               [ rgb, rgb_color,
                                 [ cmyk, cmyk_color,
                                   [ align, text_alignment ] ] ] ] ] ) )
```

Parameters

Name	Type	Description	Required
text	string	The text that should be inserted.	Yes.
font	string	The font type.	No.
size	integer	The size of the font.	No.
rgb	array	The RGB text color.	No.
cmyk	array	The CMYK text color.	No.
align	text	The text alignment.	No.

Description

This function inserts regular text into the PDF document. The font type, font size, font color, and text alignment can be specified. If not specified, this function uses the current settings. The available font types are:

- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- Courier
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Obliqu
- Helvetica
- Symbol

- Times-Bold
- Times-BoldItalic
- Times-Italic
- Times-Roman

The "rgb" and "cmyk" parameters can be used to specify the font color. Only one of these two parameters can be used at the same time. The "rgb" parameter must be an array of three integers between 0 and 255. The "cmyk" parameter must be an array of four decimal values between 0.0 and 1.0.

The alignment parameter can be used to set the text alignment, possible values are:

- left
- right
- center
- full

The last options will spread/justify the text from the left to the right margin if it covers at least 80% of the margin-width.

Examples

Example 1

```
{pdf( 'text', 'eZ publish and PDF'|wash( 'pdf' ),
      hash( 'font', 'Times-Bold', 'size', 22, 'align', 'center' ) )}
```

This example generates a centered text using "Times-Bold" at 22 points.

Example 2

```
{pdf( 'text', 'red'|wash( 'pdf' ), hash( 'rgb', array( 255, 0, 0 ) ) )
{pdf( 'text', 'green'|wash( 'pdf' ), hash( 'rgb', array( 0, 255, 0 ) ) )
{pdf( 'text', 'blue'|wash( 'pdf' ), hash( 'rgb', array( 0, 0, 255 ) ) )}
```

This example demonstrates text coloring.

Example 3

```
{pdf( 'text', 'A primary color is a color that can not be created by mixing  
other colors in the gamut of a given color space. Primary colors may  
themselves be mixed to produce most of the colors in a given color  
space.' |wash( 'pdf' ), hash( 'align', 'full' ) )}
```

This example demonstrates how the text can be justified.

5.12.23 text_box

Summary

Inserts text at a specified location.

Usage

```
pdf( text_box, hash( text, text
                      x,      x_offset,
                      y,      y_offset,
                      width, total_width,
                      [ align, text_alignment, ]
                      [ size,  text_size       ] ) )
```

Parameters

Name	Type	Description	Required
text	string	The text that should be inserted into the box.	Yes.
x	float	X coordinate of the text box.	Yes.
y	float	Y coordinate of the text box.	Yes.
width	float	The width of the text box.	Yes.
align	string	The alignment of the text.	No.
size	float	The font size.	No.

Description

This function creates a text box at the specified place on the current page. The font properties (type, size, and color) are not affected by the "set_font" (page 1115) function.

If the text box is positioned before any other content, the content will flow nicely over and under the text box. If other content is present before the text box is added, the text box will overlap it.

Examples

Example 1

```
{pdf( 'text_box',  hash( 'text', 'This text appears in a text box.' |wash(
'pdf' ),
                      'x', 250,
                      'y', 400,
                      'width', 30) )}
```

This example will write a sentence at coordinate (250, 400).

5.12.24 text_frame

Summary

Inserts a text with a frame.

Usage

```
pdf( text_frame, text, hash( [ frameRGB,      rgb_frame_color,      ]
                           [ frameCMYK,      cmyk_frame_color,      ]
                           [ textRGB,        rgb_text_color,        ]
                           [ textCMYK,       cmyk_text_color,       ]
                           [ fontSize,       font_size,          ]
                           [ fontName,       font_name,          ]
                           [ padding,        text_padding,        ]
                           [ leftPadding,    left_text_padding,    ]
                           [ rightPadding,   right_text_padding,   ]
                           [ topPadding,     top_text_padding,     ]
                           [ bottomPadding,  bottom_text_padding,  ]
                           [ roundEnds,     round_ends         ] ) )
```

Parameters

Name	Type	Description	Required
frameRGB	array	Frame RGB color.	No.
frameCMYK	array	Frame CMYK color.	No.
textRGB	array	Text RGB color.	No.
textCMYK	array	Text CMYK color.	No.
fontSize	integer	Font size.	No.
fontName	string	Font name.	No.
padding	integer	Padding around the text.	No.
leftPadding	integer	Padding left of the text.	No.
rightPadding	integer	Padding right of the text.	No.
topPadding	integer	Padding above the text.	No.
bottomPadding	integer	Padding below the text.	No.
roundEnds	boolean	Squared or rounded frame edges.	No.
text	string	The text that should be displayed inside the frame.	Yes.

Description

This function creates a text with a colored frame around it. The frame has the size of the text plus the amount of padding. The padding can be specified for each side of the text with the parameters "leftPadding", "rightPadding", "topPadding", and "bottomPadding". If the padding is equal for each side, the "padding" parameter can be used instead.

The "leftPadding" and "rightPadding" parameters may be set to "-1", which will result in padding that reaches the page margin.

By default the frame has squared corners. If the "roundEnds" parameter is set to TRUE, the frame will have rounded corners.

The colors of the frame and the text can be specified using the "frameRGB", "frameCMYK", "textRGB", and "textCMYK" parameters. The RGB parameters must be arrays consisting of three integers between 0 and 255. The CMYK parameters must be arrays consisting of four decimal values between 0.0 and 1.0.

The font can be changed using the "fontSize" and "fontName" parameters.

Examples

Example 1

```
{pdf(text_frame, "Test frame", hash( roundEnds, true(),
    textCMYK, array(0.89, 0.43, 0.01, 0),
    frameRGB, array(255,255,128),
    padding, 8,
    fontSize, 14))}
```

This example creates a frame (with round edges) around the sentence "Test frame".

5.12.25 toc

Summary

Inserts a generated table of contents.

Usage

```
pdf(toc, hash( contentText, toc_header_text,
                [ size,           size_array,   ]
                [ dots,           boolean,      ]
                [ indent,         indent_array ] ) )
```

Parameters

Name	Type	Description	Required
contentText	string	Table of contents header.	Yes.
size	array	An array of numbers, indicating the font sizes for each header level.	No.
indent	array	An array of booleans, indicating whether the title should be indented for each header level.	No.
dots	boolean	Display dots between the titles and page numbers.	No.

Description

This function generates and inserts the table of contents after the front page. If the front page is not available, the table of contents is inserted at the beginning of the PDF document.

Examples

Example 1

```
{pdf( 'toc', hash( 'size', array( 18, 16, 14, 12, 10 ),
                    'dots', true(),
                    'contentText', 'Content' |wash( 'pdf' ),
                    'indent', array( 0, 4, 6, 8, 10 ) ) )}
```

This example creates a table of contents. The size of the level 1 headers will be 18 and will not be indented, level 2 headers will have size 16 and will be indented 4 dots, etc. There will be dots between the headers and the page numbers.

5.12.26 ul

Summary

Inserts a bullet list.

Usage

```
pdf( ul, text, hash( [ rgb,           rgb_color,      ]
                      [ cmyk,         cmyk_color,    ]
                      [ radius,       dot_radius,   ]
                      [ indent,        text_indent,  ]
                      [ pre_indent,   bullet_indent ] ) )
```

Parameters

Name	Type	Description	Required
text	string	Bullet text.	Yes.
rgb	array	Array of RGB colors.	No.
cmyk	array	Array of CMYK colors.	No.
radius	float	The radius of the dot.	No.
indent	float	Text indentation after the dot.	No.
pre_indent	float	Indentation before the bullet.	No.

Description

This function inserts a bullet list into the PDF document.

The colors of the bullet can be specified using the "rgb" or the "cmyk" parameter. The "rgb" parameter must be an array consisting of three integers between 0 and 255. The "cmyk" parameter must be array of four decimal numbers between 0.0 and 1.0.

The size of the bullet can be controlled using the "radius" parameter. The indentation (in dots) before and after the dot can be specified using the "pre_indent" and "indent" parameters.

Examples

Example 1

```
{pdf( 'text', "Most popular internet browsers in 2004:" |wash( 'pdf' )}

{pdf( 'ul', 'Internet Explorer (88,9%)' |wash( 'pdf' ) )
{pdf( 'ul', 'Version 6 (80.95%)' |wash( 'pdf' ), hash( 'pre_indent', 15 ) )
{pdf( 'ul', 'Version 5.5 (4.18%)' |wash( 'pdf' ), hash( 'pre_indent', 15 ) )
{pdf( 'ul', 'Version 5.0 (3.66%)' |wash( 'pdf' ), hash( 'pre_indent', 15 ) )}
```

```
{pdf( 'ul', 'Mozilla-based browsers (7.35%)' | wash( 'pdf' ) )}  
{pdf( 'ul', 'Rest (3.75%)' | wash( 'pdf' ) )}
```

This example generates a bullet list where some of the bullets are indented.

5.13 Configuration files

binaryfile.ini (page 1135)

Not documented yet.

browse.ini (page 1136)

Not documented yet.

collaboration.ini (page 1137)

Not documented yet.

collect.ini (page 1138)

Not documented yet.

content.ini (page 1139)

Settings related to content.

contentstructuremenu.ini (page 1161)

Not documented yet.

cronjob.ini (page 1162)

Settings related to cronjobs.

datatype.ini (page 1171)

Not documented yet.

datetime.ini (page 1172)

Not documented yet.

dbschema.ini (page 1173)

Not documented yet.

debug.ini (page 1174)

Not documented yet.

design.ini (page 1175)

Settings related to designs and design related files like css and javascripts.

error.ini (page 1184)

Not documented yet.

extendedattributefilter.ini (page 1185)

Not documented yet.

ezxml.ini (page 1186)

Not documented yet.

fetchalias.ini (page 1187)

Not documented yet.

file.ini (page 1188)

Not documented yet.

i18n.ini (page 1189)

Settings related to internationalization.

icon.ini (page 1192)

Not documented yet.

image.ini (page 1193)

Not documented yet.

layout.ini (page 1194)

Not documented yet.

ldap.ini (page 1195)

Not documented yet.

logfile.ini (page 1197)

Not documented yet.

menu.ini (page 1198)

Not documented yet.

module.ini (page 1199)

Not documented yet.

notification.ini (page 1200)

Not documented yet.

override.ini (page 1201)

Not documented yet.

package.ini (page 1202)

Not documented yet.

paymentgateways.ini (page 1203)

Not documented yet.

setup.ini (page 1204)

Not documented yet.

shopaccount.ini (page 1205)

Not documented yet.

site.ini (page 1206)

Controls the overall/main behavior of the system.

soap.ini (page 1421)

Not documented yet.

staticcache.ini (page 1422)

Settings related to the static cache.

template.ini (page 1429)

Not documented yet.

textfield.ini (page 1430)

Not documented yet.

texttoimage.ini (page 1431)

Not documented yet.

toolbar.ini (page 1432)

Not documented yet.

transform.ini (page 1433)

Not documented yet.

units.ini (page 1434)

Not documented yet.

upload.ini (page 1435)

Not documented yet.

viewcache.ini (page 1436)

Not documented yet.

webdav.ini (page 1437)

Not documented yet.

wordtoimage.ini (page 1438)

Not documented yet.

workflow.ini (page 1439)

Settings related to workflows.

5.13.1 binaryfile.ini

5.13.2 browse.ini

5.13.3 collaboration.ini

5.13.4 collect.ini

5.13.5 content.ini

The configuration blocks are documented in the following sections:

- [ActionSettings] (page [1146](#))
- [DataTypeSettings] (page [1148](#))
- [HideSettings] (page [1143](#))
- [UnpublishSettings] (page [1140](#))
- [VersionManagement] (page [1152](#))
- [VersionView] (page [1156](#))

[UnpublishSettings]**ClassList** ([page 1141](#))

Sets the content classes that use the "Date and time" datatype (used by the "unpublish.php" cronjob).

RootNodeList ([page 1142](#))

Controls which subtrees will be affected by the "unpublish.php" cronjob.

ClassList

Summary

Sets the content classes that use the "Date and time" datatype (used by the "unpublish.php" cronjob).

Usage

```
ClassList []=class_id1  
ClassList []=class_id2  
...
```

Description

This setting reveals the content classes that contain attributes which can be used by the "unpublish.php" script. Please note that you'll have to use ID numbers here, not identifiers. Only objects that are of these classes will be affected by the script. In addition, the "RootNodeList (page 1142)" directive must be used to specify which subtrees that can have their nodes affected by the cronjob script.

Examples

Example 1

```
ClassList []=2
```

Assuming that the ID number of your article class is 2, this configuration will tell the "unpublish.php" cronjob to check the date and time values specified in the "unpublish_date" attribute of the article(s).

RootNodeList

Summary

Controls which subtrees that will be affected by the "unpublish.php" cronjob.

Usage

```
RootNodeList []=node_id1  
RootNodeList []=node_id2  
...
```

Description

If you are using the "unpublish.php" cronjob to delete (move to Trash) your content objects when a specified date and time is reached, you can use this directive to specify which parent nodes that will have their children (recursively) affected by the unpublish feature. The "RootNodeList" configuration array is empty by default, i.e. no objects will be removed.

Examples

Example 1

```
RootNodeList []=2
```

This will tell the system that the "unpublish.php" cronjob can be applied to any item in the content tree.

[HideSettings]**HideDateAttributeList** (page 1144)

Sets which date/time attributes the hide cronjob should use.

RootNodeList (page 1145)

Sets which subtrees that will be affected by the "hide.php" cronjob.

HideDateAttributeList

Summary

Sets which date/time attributes the hide cronjob should use.

Usage

```
HideDateAttributeList[class_id1]=attribute_id1  
HideDateAttributeList[class_id2]=attribute_id2  
...
```

Description

This directive can be used to tell the "hide.php" cronjob about which date/time values it should look for when hiding nodes. Both the class and the attribute (which needs to be represented by the "Date and time (page 276)" datatype) must be specified using identifiers (not ID numbers). In addition, the "RootNodeList (page 1145)" directive must be used to specify which subtrees that can have their nodes affected by the cronjob script.

Examples

Example 1

```
HideDateAttributeList[article]=hide_date
```

The hide cronjob will check date/time values specified in the "hide_date" attribute of your articles.

RootNodeList

Summary

Sets which subtrees that will be affected by the "hide.php" cronjob.

Usage

```
RootNodeList []=node_id1  
RootNodeList []=node_id2  
...
```

Description

If you are using the "hide.php" cronjob to control the visibility of your nodes (based on a date/time-attribute), you must use this directive to specify which subtrees that can have their children (recursively) affected by this feature. Please note that the default value is empty, which means that no nodes will be hidden.

Examples

Example 1

```
RootNodeList []=2
```

This tells the system that the "hide.php" cronjob can affect all items in the content node tree.

[ActionSettings]**ExtensionDirectories** (page 1147)

Sets the directories where eZ Publish will look for action extensions.

ExtensionDirectories

Summary

Sets the directories where eZ Publish will look for action extensions.

Usage

```
ExtensionDirectories []= directory1
ExtensionDirectories []= directory2
...
...
```

Description

eZ Publish will look for actions in ” ExtensionDirectory (page 1257)/ *directory/actions/content_actionhandler.php*”. In that file eZ Publish will look for the function *directory_ContentActionHandler*. This function will be invoked when your action is triggered. The function must accept three parameters: ”module”, ”http” and ”objectID”.

[DataTypeSettings]**AvailableDatatypes** (page 1149)

Sets the datatypes available to eZ Publish.

ExtensionDirectories (page 1150)

Sets the extensions that contains datatypes.

RepositoryDirectories (page 1151)

Sets the directories where eZ Publish should look for datatypes.

AvailableDatatypes

Summary

Sets the datatypes available to eZ Publish.

Usage

```
AvailableDatatypes []= datatype1  
AvailableDatatypes []= datatype2  
...
```

Description

eZ Publish will look for the datatypes in the directories specified by RepositoryDirectories (page 1151) and ExtensionDirectories (page 1150).

Examples

```
ExtensionDirectory []=my_extension  
AvailableDatatypes []=my_datatype
```

This example shows a typical configuration of content.ini.append in an extension with a datatype. If extensions are located in the "extension" directory (default) these settings will make eZ Publish look for the datatype extension in the directory "/extension/my_extension/datatypes/my_datatype/".

ExtensionDirectories

Summary

Sets the extensions that contains datatypes.

Usage

```
ExtensionDirectories []= directory1
ExtensionDirectories []= directory2
...
...
```

Description

eZ Publish will look for datatypes in ” ExtensionDirectory (page 1257)/ *directory/datatypes/*”. You must also add your datatype to AvailableDataTypes (page 1149) in order to make eZ Publish recognize it.

RepositoryDirectories

Summary

Sets the directories where eZ Publish should look for datatypes.

Usage

```
RepositoryDirectories[] = directory1
RepositoryDirectories[] = directory2
```

Description

Directories should be specified relative to your eZ Publish root directory.

This setting is for kernel datatypes only. If you add your own datatype you should put it in an extension and use the ExtensionDirectories (page 1150) setting to tell eZ Publish where to find it.

[VersionManagement]

Read more about the versioning system in eZ Publish.

DefaultVersionHistoryLimit ([page 1155](#))

Sets the number of concurrent versions that can exist of an object.

DeleteDrafts ([page 1153](#))

Sets if drafts should be discarded if there is no room to create a new draft.

VersionHistoryClass ([page 1154](#))

Sets the number of concurrent versions that can exist of an object per class.

DeleteDrafts

Summary

Sets if drafts should be discarded if there is no room to create a new draft.

Usage

```
DeleteDrafts=enabled|disabled
```

Description

This setting controls what eZ Publish should do if a user wants to create a new draft but there are no "empty" versions.

- **disabled** - (default) The oldest archived version is discarded. If there are no archived versions to discard the user is shown an error screen and is given the possibility to remove drafts.
- **enabled** - The oldest archived version is discarded. If there are no archived versions to discard eZ Publish discards the oldest draft.

VersionHistoryClass

Summary

Sets the number of concurrent versions that can exist of an object per class.

Usage

```
VersionHistoryClass[ class_id1 ]= number  
VersionHistoryClass[ class_id2 ]= number  
...
```

Description

This setting is similar to DefaultVersionHistoryLimit (page 1155) but allows you to set the maximum number of versions per class giving you more fine grained control.

A typical scenario is to limit the number of versions of posts in your public forum, but not for the other classes in the system.

Examples

```
VersionHistoryClass[1]=5  
VersionHistoryClass[3]=3
```

Using these settings eZ Publish will allow 5 versions for objects of classes with id 1 and 3 versions for objects of class 3. Other objects in your system will use the global DefaultVersionHistoryLimit (page 1155) to determine the maximum number of versions.

DefaultVersionHistoryLimit**Summary**

Sets the number of concurrent versions that can exist of an object.

Usage

```
DefaultVersionHistoryLimit= number
```

Description

This setting globally determines how many versions eZ Publish will store of a specific object including drafts.

You should never set this setting below 2 since you need to allow at least one published version and one draft.

[VersionView]**AllowChangeButtons** (page 1158)

This setting is not used anymore.

AllowVersionsButton (page 1157)

This setting is not used anymore.

AvailableSiteDesignList (page 1160)

Sets the sitedesigns that are used by the complete site.

DefaultPreviewDesign (page 1159)

Sets the default preview design for eZ Publish releases prior to 3.6.

AllowVersionsButton**Summary**

This setting is not used anymore.

Usage

```
AllowVersionsButton=enabled|disabled
```

AllowChangeButtons**Summary**

This setting is not used anymore.

Usage

```
AllowChangeButtons=enabled|disabled
```

DefaultPreviewDesign**Summary**

Sets the default preview design for eZ Publish releases prior to 3.6.

Usage

```
DefaultPreviewDesign= design_name
```

AvailableSiteDesignList

Summary

Sets the sitedesigns that are used by the complete site.

Usage

```
AvailableSiteDesignList []= design1
AvailableSiteDesignList []= design2
...
...
```

Description

This setting defines the complete set of designs (page 97) that is used by your site. The value is used by the caching system to clear the correct caches when your site is updated. In addition the designs listed here will be available when you are previewing content.

eZ Publish will only clear the caches of the listed designs when content is published. To ensure the correct operation of your site it is important that you list all the designs you are using with this setting.

Examples

```
AvailableSiteDesignList []= standard
AvailableSiteDesignList []= base
AvailableSiteDesignList []= admin
```

5.13.6 contentstructuremenu.ini

5.13.7 cronjob.ini

The configuration blocks are documented in the following sections:

- [CronjobPart-CRONJOB_TASK] (page [1163](#))
- [CronjobSettings] (page [1167](#))
- [linkCheckSettings] (page [1165](#))

[CronjobPart-CRONJOB_TASK]

These cronjob tasks can easily (separately) be scheduled. The configuration settings in this group define which scripts must be run.

The first parameter after the runcronjob.php script defines the cronjob_task.

Scripts (page 1164)

Specifies the cronjob scripts that will be executed for this cronjob part.

Scripts

Summary

Specifies the cronjob scripts that will be executed for this cronjob part.

Usage

```
Scripts[] = cronjob_script1  
Scripts[] = cronjob_script2  
...
```

Examples

```
[CronjobPart-allWorkflows]  
Scripts[] = workflow1.php  
Scripts[] = workflow2.php
```

If the cronjob part "allWorkflows" is called, the scripts workflow1 and workflow2 will be executed.

[linkCheckSettings]**SiteURL** (page 1166)

Sets the site URLs to use when validating internal links by the linkcheck cronjob.

SiteURL

Summary

Sets the site URLs to use when validating internal links by the linkcheck cronjob.

Usage

```
SiteURL []=URL1  
SiteURL []=URL2  
...
```

Description

Specify your site URLs so that the "linkcheck.php" cronjob will handle relative URLs (internal links) properly.

Examples

```
SiteURL []=http://admin.mysite.com  
SiteURL []=http://mysite.com
```

This example will tell the linkcheck script to use "http://admin.mysite.com" and "http://mysite.com" site URLs when checking relative URLs like "/products/black_box".

[CronjobSettings]**ExtensionDirectories** (page 1168)

Sets one or multiple extension directories that may contain cronjobs.

ScriptDirectories (page 1170)

Sets one or multiple directories that may contain cronjobs.

Scripts (page 1169)

Specifies the cronjob scripts that will be executed.

ExtensionDirectories

Summary

Sets one or multiple extension directories that may contain cronjobs.

Usage

```
ExtensionDirectories []= extension1
ExtensionDirectories []= extension2
...
...
```

Description

The cronjob script will search through all the "cronjobs" directories of the specified extensions.

Examples

```
Scripts []=workflow.php
Scripts []=notification.php
Scripts []=linkcheck.php
ExtensionDirectories []=myExtension
```

This example will search through the myExtension/cronjobs directories to find the workflow, notification, and linkcheck cronjobs.

Scripts

Summary

Specifies the cronjob scripts that will be executed.

Usage

```
Scripts[] = cronjob_script
Scripts[] = cronjob_script2
...
...
```

Examples

```
ScriptDirectories[] = cronjobs
ScriptDirectories[] = cronjobs2
Scripts[] = workflow.php
Scripts[] = notification.php
Scripts[] = linkcheck.php
```

This example will search through the cronjobs and cronjobs2 directories to find the workflow, notification, and linkcheck cronjobs.

ScriptDirectories

Summary

Sets one or multiple directories that may contain cronjobs.

Usage

```
ScriptDirectories[] = directory
ScriptDirectories[] = directory2
...
...
```

Description

The cronjob script will search through all the ScriptDirectories to find cronjobs.

Examples

```
ScriptDirectories[] = cronjobs
ScriptDirectories[] = cronjobs2
Scripts[] = workflow.php
Scripts[] = notification.php
Scripts[] = linkcheck.php
```

This example will search through the cronjobs and cronjobs2 directories to find the workflow, notification, and linkcheck cronjobs.

5.13.8 datatype.ini

5.13.9 datetime.ini

5.13.10 dbschema.ini

5.13.11 debug.ini

5.13.12 design.ini

The configuration blocks are documented in the following sections:

- [ExtensionSettings] (page [1182](#))
- [JavaScriptSettings] (page [1176](#))
- [StyleSheetSettings] (page [1178](#))

[JavaScriptSettings]**JavaScriptList** (page 1177)

Sets the Javascript files to include in the pagelayout.

JavaScriptList

Summary

Sets the Javascript files to include in the pagelayout.

Usage

```
JavaScriptList []= javascript1  
JavaScriptList []= javascript2  
...
```

Description

You should provide the full path relative to the design directory.

Examples

```
JavaScriptList []=javascripts/mynewticker.js
```

Using this configuration eZ Publish will load the file "/design/ mydesign/javascripts/ mynewsticker.js"

[StyleSheetSettings]**ClassesCSS** (page 1180)

Sets the CSS files which contains class definitions for the base layout.

CSSFileList (page 1179)

Sets the CSS files to include in the pagelayout.

SiteCSS (page 1181)

Sets the global CSS file for designs based on the base layout.

CSSFileList

Summary

Sets the CSS files to include in the pagelayout.

Usage

```
CSSFileList []= css_file1  
CSSFileList []= css_file2  
...
```

Description

Sets the CSS files that should automatically be included in the pagelayout. You should provide the full path relative to the stylesheets subdirectory in the design directory.

Examples

```
CSSFileList []=mycss.css
```

Using this configuration eZ Publish will load the file ”/design/ *mydesign*/stylesheets/mycss.css”.

ClassesCSS**Summary**

Sets the CSS files which contains class definitions for the base layout.

Usage

```
ClassesCSS= path_to_CSS_file
```

SiteCSS**Summary**

Sets the global CSS file for designs based on the base layout.

Usage

```
SiteCSS= path_to_css_file
```

[ExtensionSettings]**DesignExtensions** (page 1183)

Sets which extensions that have designs.

DesignExtensions

Summary

Sets which extensions that have designs.

Usage

```
DesignExtensions[] = extension1  
DesignExtensions[] = extension2  
...
```

Description

By default eZ Publish will only look for designs in the ”/design” directory. If your extensions provide designs you must tell eZ Publish by providing the extension name in the DesignExtensions setting.

eZ Publish will look for designs in the directory ”/extension/ *extension_name/design/*”

This setting is typically overriden in the design.ini.append in extensions.

Examples

```
DesignExtensions[] = myextension
```

eZ Publish will now search for designs in ”/extension/myextension/design”.

5.13.13 error.ini

5.13.14 extendedattributefilter.ini

5.13.15 ezxml.ini

5.13.16 fetchalias.ini

5.13.17 file.ini

5.13.18 i18n.ini

The configuration blocks are documented in the following sections:

- [CharacterSettings] (page [1190](#))

[CharacterSettings]**Charset** (page 1191)

Sets the character set that eZ Publish should use

Charset

Summary

Sets the character set that eZ Publish should use

Usage

```
Charset=character_set
```

Description

Use this directive to set the internal charset for the site. Specify UTF-8 for multilingual sites or a specific charset for monolingual sites.

Examples

Example 1

```
Charset=utf-8
```

This will set the output character set of eZ Publish to UTF-8.

5.13.19 icon.ini

5.13.20 image.ini

5.13.21 layout.ini

5.13.22 ldap.ini

The configuration blocks are documented in the following sections:

- [LDAPSettings] (page [1196](#))

[LDAPSettings]

5.13.23 logfile.ini

5.13.24 menu.ini

5.13.25 module.ini

5.13.26 notification.ini

5.13.27 override.ini

Please refer to the following pages:

- The template override system (page [222](#))
- Template override example (page [224](#))
- Template override conditions (page [1065](#))

5.13.28 package.ini

5.13.29 paymentgateways.ini

5.13.30 setup.ini

5.13.31 shopaccount.ini

5.13.32 site.ini

The configuration blocks are documented in the following sections:

- [ContentSettings] (page [1207](#))
- [DatabaseSettings] (page [1219](#))
- [DebugSettings] (page [1241](#))
- [DesignSettings] (page [1250](#))
- [ExtensionSettings] (page [1254](#))
- [FileSettings] (page [1258](#))
- [FormProcessSettings] (page [1268](#))
- [InformationCollectionSettings] (page [1270](#))
- [MailSettings] (page [1272](#))
- [OverrideSettings] (page [1284](#))
- [PortAccessSettings] (page [1286](#))
- [RegionalSettings] (page [1288](#))
- [RoleSettings] (page [1300](#))
- [RSSSettings] (page [1305](#))
- [SearchSettings] (page [1311](#))
- [Session] (page [1322](#))
- [SetupSettings] (page [1329](#))
- [ShopSettings] (page [1334](#))
- [SiteAccessRules] (page [1337](#))
- [SiteAccessSettings] (page [1340](#))
- [SiteSettings] (page [1366](#))
- [TemplateSettings] (page [1375](#))
- [TipAFriend] (page [1388](#))
- [UnitSettings] (page [1392](#))
- [URLTranslator] (page [1395](#))
- [UserSettings] (page [1399](#))

[ContentSettings]**CacheDir** (page 1218)

Sets the directory where eZ publish stores view cache files.

CachedViewModes (page 1216)

Sets which of the content view modes that use view cache.

CachedViewPreferences (page 1215)

Sets the user preferences each view mode depends on.

CacheThreshold (page 1212)

Sets the threshold for content cache cleanup

ComplexDisplayViewModes (page 1213)

Sets content views that have their content cache expired whenever an object is published.

EditDirtyObjectAction (page 1211)

Sets what eZ Publish should do if the user tries to edit a page that has a draft that is newer than the published version.

PreCacheSiteaccessArray (page 1209)

Sets the siteaccesses that will have view-cache created when an object is published.

PreViewCache (page 1210)

Sets if eZ publish should generate the view cache when an object is published.

PreviewCacheUsers (page 1208)

Sets the users that should have view cache generated when an object is published.

StaticCache (page 1214)

Sets if static caching should be enabled or not.

ViewCaching (page 1217)

Sets if view caching should be enabled or not.

PreviewCacheUsers

Summary

Sets the users that should have view cache generated when an object is published.

Usage

```
PreviewCacheUsers []=anonymous|current|user_id1  
PreviewCacheUsers []=anonymous|current|user_id2  
...
```

Description

By default eZ publish will generate view cache for the anonymous user. Using this setting you can specify additional user IDs that eZ publish will generate view cache for. You must enable PreViewCache (page [1210](#)) for this setting to have any effect.

Examples

```
PreviewCacheUsers []=anonymous  
PreviewCacheUsers []=current  
PreviewCacheUsers []=23
```

Using these settings eZ publish will generate view cache for the anonymous user, the current user and the user with ID 23 when an object is published.

PreCacheSiteaccessArray

Summary

Sets the siteaccesses that will have view-cache created when an object is published.

Usage

```
PreCacheSiteaccessArray[] = siteaccess1  
PreCacheSiteaccessArray[] = siteaccess2  
...
```

Description

You must enable PreViewCache (page [1210](#)) for this setting to have any effect.

Examples

```
PreviewCache=enabled  
PreCacheSiteaccessArray[] = admin  
PreCacheSiteaccessArray[] = base
```

With these settings eZ publish will create view cache when an object is published for the admin and the base siteacesses.

PreViewCache

Summary

Sets if eZ publish should generate the view cache when an object is published.

Usage

```
PreViewCache=enabled|disabled
```

Description

Enabling this setting will make the publishing process a bit slower. However, the first request to the page will be a bit faster. For sites with lots of content editors you should disable this setting.

EditDirtyObjectAction

Summary

Sets what eZ Publish should do if the user tries to edit a page that has a draft that is newer than the published version.

Usage

```
EditDirtyObjectAction=showversions|usecurrent
```

Description

If you select *showversions* eZ Publish will display a page with all the available versions, including drafts. The user can then select which version he wants to base the new draft on.

If you select *usecurrent* eZ Publish will create a new draft based on the published version and let the user edit that.

CacheThreshold**Summary**

Sets the threshold for content cache cleanup

Usage

```
CacheThreshold= number
```

Description

The threshold for file cleanup, if it is exceeded a global expiry is used instead. The value is calculated with the number of affected nodes * viewmodes * translations * sitedesign.

Note: This is an internal change that you should not change unless you are a developer.

ComplexDisplayViewModes

Summary

Sets content views that have their content cache expired whenever an object is published.

Usage

```
ComplexDisplayViewModes= view1 [; view2] ...
```

StaticCache

Summary

Sets if static caching should be enabled or not.

Usage

```
StaticCache=enabled|disabled
```

Description

Static caching means that some pages on your system will be stored and served completely in HTML with a huge speed improvement. You can only use static caching for pages that do not have any dynamic elements (e.g the pages are available to everyone and looks exactly the same to all users).

Static caching is configured in staticcache.ini (page [1422](#)).

CachedViewPreferences

Summary

Sets the user preferences each view mode depends on.

Usage

```
CachedViewPreferences[ viewmode1]= setting1 [= defaultValue1]; setting2 [= defaultValue2]...
CachedViewPreferences[ viewmode2]= setting1 [= defaultValue1] ; setting2 [= defaultValue2]...
...
```

Description

Sometimes you have conditions based on user preferences (page 810) in the templates of cached view modes (page 1216). eZ publish needs to know the preferences you use per view mode in order to make sure that the caches are correct. You should use this setting if at least one of your templates (including override templates) use a condition based on a user preference.

A typical symptom of a missing CachedViewPreferences setting is if you change a preference setting and the interface is not updated until you clear the cache.

Examples

```
CachedViewPreferences[full]=mysetting;myothersetting=1
CachedViewPreferences[pdf]=myothersetting=1
```

This example shows a site where at least one of the templates of the "full" view mode uses the mysetting and myothersetting. One of the templates in the "pdf" view also uses the myothersetting setting.

CachedViewModes

Summary

Sets which of the content view modes that use view cache.

Usage

```
CachedViewModes=view1[;view2][;view3]...
```

Description

This setting enables the viewcache for the specified view modes inside the "view" view of the content module. When view caching is enabled, the entire result of the module will be cached. The cache is stored for each possible role combination on your site. This means that your templates can have conditions based on roles even when caching is on.

In addition, you can enable view caching for the "pdf" view of the content module by specifying "pdf" in this setting. In this case, the actual PDF file will be cached.

Note:Do not change this setting unless you know what you are doing.

Examples

```
CachedViewModes=full;sitemap;pdf
```

This makes eZ publish use view cache on the "full" and "sitemap" content view modes ("content/view/full" and "content/view/sitemap"). The output of the "pdf" view within the content module ("content/pdf") will also be cached.

ViewCaching

Summary

Sets if view caching should be enabled or not.

Usage

```
ViewCaching=enabled|disabled
```

Description

Viewcache is the terminology we use for cache that stores the complete HTML output of a view (page [149](#)).

You can turn off view cache during development of a site to force eZ publish to render all templates on each request.

Note: Live sites should always have ViewCache enabled.

CacheDir

Summary

Sets the directory where eZ publish stores view cache files.

Usage

```
CacheDir= directory_name
```

Description

CacheDir is set relative to CacheDir (page 1260). This means that view cache files are stored within `/var_dir/cache_dir/directory_name` where var_dir is specified by the VarDir (page 1261) setting and cache_dir is specified by the CacheDir (page 1260) setting.

Examples

```
[FileSettings]
VarDir=var
CacheDir=cache

[ContentSettings]
CacheDir=content
```

Using these settings the viewcache will be stored in ”/var/cache/content”

[DatabaseSettings]**Charset** (page 1234)

Sets the character set that eZ publish uses when communicating with the database.

ConnectRetries (page 1236)

Sets the number of database connection retries.

Database (page 1238)

Sets the database to use when connecting to the database server.

DatabaseImplementation (page 1222)

Sets the type of database you are using.

DatabasePluginPath (page 1221)

Sets the path to an external database driver.

Host (page 1237)

Sets the host that contains the database eZ publish should use.

ImplementationAlias (page 1228)

Sets alias names for database implementations

Password (page 1239)

Sets the password that eZ publish uses when logging in to the database.

SlaverServerDatabase (page 1223)

The databases to use when logging in to the slaveservers.

SlaverServerPassword (page 1224)

The passwords to use when logging in to the slaveservers.

SlaverServerUser (page 1225)

The usernames to use when logging in to the slaveservers.

SlaveServerArray (page 1226)

The hostnames of the slaveservers to use for read queries.

SlowQueriesOutput (page 1230)

Show queries that where slower than a set amount of time.

Socket (page 1232)

Sets the socket eZ publish should use when connecting to the database.

SQLOutput (page 1231)

Enables the output of SQL queries in the debug output.

Transactions (page 1235)

Enables to make eZ publish use transactions to ensure database integrity.

UseBuiltInEncoding (page 1233)

Use the built in character conversion in the database if available.

UsePersistentConnection (page 1229)

Controls if database connections should be kept open between eZ publish runs.

User (page 1240)

Sets the username that eZ publish uses when logging in to the database.

UseSlaveServer (page 1227)

Enables the usage of slave database servers for read queries. (MySQL only)

DatabasePluginPath

Summary

Sets the path to an external database driver.

Usage

```
DatabasePluginPath= path_to_database_driver
```

Description

If you want to use a custom database driver you must tell eZ publish where to find it. Use the path relative to the root of your eZ publish installation. eZ publish will search for the file "dbname" + "db.php" in that directory.

Examples

```
DatabaseImplementation=custom  
DatabasePluginPath=extensions/mydbdriver/classes/
```

eZ publish will now search for the file customdb.php in the directory extensions/mydbdriver/classes/

DatabaseImplementation

Summary

Sets the type of database you are using.

Usage

```
DatabaseImplementation=ezmysql|ezpostgresql
```

Description

Set this option to ezmysql if you are using a MySQL database. If you are using PostgreSQL set this option to ezpostgresql.

SlaverServerDatabase

Summary

The databases to use when logging in to the slaveservers.

Usage

```
SlaveServerDatabase []
SlaveServerDatabase []= database1
SlaveServerDatabase []= database2
...
```

Description

Specify one database name on each row of the setting. The order is significant and you have to use the same order for the settings in SlaveServerArray, SlaverServerUser and SlaverServerPassword

Examples

```
SlaverServerDatabase []
SlaverServerDatabase []=publishslave
SlaverServerDatabase []=publishslave2
```

This setup will use the database publishslave in the first slave server and the database publishslave2 in the second slave server.

SlaverServerPassword

Summary

The passwords to use when logging in to the slaveservers.

Usage

```
SlaverServerPassword []
SlaverServerPassword []= password1
SlaverServerPassword []= password2
```

Description

Specify one password on each row of the setting. The order is significant and you have to use the same order for the settings in SlaveServerArray, SlaverServerUser and SlaverServerDatabase.

Examples

```
SlaverServerPassword []
SlaverServerPassword []=secret
SlaverServerPassword []=verysecret
```

This setup will use the password secret to log in to the first slave server and the password very-secret to log in to the second slave server.

SlaverServerUser

Summary

The usernames to use when logging in to the slaveservers.

Usage

```
SlaveServerUser []
SlaveServerUser []= user1
SlaveServerUser []= user2
```

Description

Specify one username on each row of the setting. The order is significant and you have to use the same order for the settings in SlaveServerArray, SlaverServerPassword and SlaverServerDatabase

Examples

```
SlaverServerUser []
SlaverServerUser []=admin
SlaverServerUser []=root
```

This setup will use the username admin to log in to the first slave server and the username root to log in to the second slave server.

SlaveServerArray

Summary

The hostnames of the slaveservers to use for read queries.

Usage

```
SlaveServerArray[]  
SlaveServerArray[] = hostname1  
SlaveServerArray[] = hostname2  
...
```

Description

Specify one database server on each row of the setting. The order is significant and you have to use the same order for the settings in SlaverServerUser, SlaverServerPassword and SlaverServerDatabase.

Examples

```
SlaveServerArray[]  
SlaveServerArray[] = donald  
SlaveServerArray[] = mickey
```

This setup will load balance read queries between the main database server and the database servers located on donald and mickey.

UseSlaveServer

Summary

Enables the usage of slave database servers for read queries. (MySQL only)

Usage

```
UseSlaveServer=enabled|disabled
```

Description

Set this option to enabled to make eZ publish load balance read queries between several SQL servers. This option is database independent and works for all database implementations. You can set the location and login information for the slave servers using the settings:

- SlaveServerArray
- SlaverServerUser
- SlaverServerPassword
- SlaverServerDatabase

For all other connection options the values set for the standard database will be used.

This option only works on MySQL setups. You can read more about how to set up MySQL replication here.

ImplementationAlias

Summary

Sets alias names for database implementations

Usage

```
ImplementationAlias [ aliasname ]=ezpostgresql|ezmysql  
....
```

Description

The various database implementations in eZ publish uses a database driver. These database drivers have names like ezmysql and ezpostgresql. In order to make let users use more commonly known names like mysql and postgresql you can set up aliases for the database drivers.

Don't change this setting unless you know what you are doing.

Examples

```
ImplementationAlias []  
ImplementationAlias [mysql]=ezmysql
```

This setting means that you can use the alias mysql when you want to use the ezmysql driver.

UsePersistentConnection

Summary

Controls if database connections should be kept open between eZ publish runs.

Usage

```
UsePersistentConnection=enabled|disabled
```

Description

Persistent connections are usually faster but have had a lot of unwanted effects in the past. Don't turn this on unless you know what you are doing.

SlowQueriesOutput

Summary

Show queries that where slower than a set amount of time.

Usage

```
SlowQueriesOutput= number
```

Description

The number controls how many milliseconds a query must take in order to be displayed in the debug output. 0 means that all queries will be shown.

SQLOutput

Summary

Enables the output of SQL queries in the debug output.

Usage

```
SQLOutput=enabled|disabled
```

Description

Debug (page [1385](#)) must be enabled for this setting to have any effect.

Socket**Summary**

Sets the socket eZ publish should use when connecting to the database.

Usage

```
Socket= number|disabled
```

Description

Set this option to 0 to use the default socket for the selected database.

UseBuiltInEncoding

Summary

Use the built in character conversion in the database if available.

Usage

```
UseBuiltInEncoding=true|false
```

Description

If this option is set to false or if the conversion is not available in the database the conversion will be performed by eZ publish at a somewhat slower speed.

Charset**Summary**

Sets the character set that eZ publish uses when communicating with the database.

Usage

```
Charset= charset
```

Description

If this setting is left empty the setting from i18n.ini will be used. Usually this setting is sufficient.

Transactions

Summary

Enables to make eZ publish use transactions to ensure database integrity.

Usage

```
Transactions=enabled|disabled
```

Description

If you enable this setting eZ publish will perform all queries that naturally belong together within a transaction. This ensure that your database integrity will be kept even if the webserver crashes in the middle of the transaction. Don't turn this off unless you know what you are doing.

MySQL requires tables of the type InnoDB in order to use transactions.

ConnectRetries

Summary

Sets the number of database connection retries.

Usage

```
ConnectRetries= number
```

Description

This directive can be used to set the number of times eZ Publish should attempt to connect to the database if the initial attempt fails. Note that the number of actual connection attempts equals the initial attempt plus the number of retries specified by this directive. In other words, if the number of connection retries is set to 3, eZ Publish will attempt to connect to the database 4 times before giving up and generating an error message.

Examples

```
ConnectRetries=3
```

This setting makes eZ publish attempt to connect to the database four times in total. One main attempt and three retries.

Host**Summary**

Sets the host that contains the database eZ publish should use.

Usage

```
Host= hostname
```

Description

You can use both hostnames and IP addresses. Use "localhost" if the database and webserver are located on the same machine.

Database**Summary**

Sets the database to use when connecting to the database server.

Usage

```
Database= databasename
```

Password**Summary**

Sets the password that eZ publish uses when logging in to the database.

Usage

```
User= password
```

User**Summary**

Sets the username that eZ publish uses when logging in to the database.

Usage

```
User= username
```

[DebugSettings]**Debug** (page 1245)

Sets if you want to display debug information in the rendered page or in a separate popup.

DebugByIP (page 1247)

Enables debug output for some IP addresses only. Useful when debugging live sites.

DebugIPList (page 1246)

Sets the hosts that receive debug output.

DebugLogOnly (page 1242)

Choose if you want debug strings in the debugoutput or in the log only.

DebugOutput (page 1249)

Main switch for debug output

DebugRedirection (page 1244)

Enables debugging of internal and external module redirections.

DisplayDebugWarnings (page 1243)

Choose if debug warnings should be displayed explicitly on the top of the page or in the debug log only.

ScriptDebugOutput (page 1248)

Enables debug output for PHP scripts run from the command line.

DebugLogOnly

Summary

Choose if you want debug strings in the debugoutput or in the log only.

Usage

```
DebugLogOnly=enabled|disabled
```

Description

Enable this option to remove debug strings from the inline (or popup) debug output. Instead the debug strings will only be appended to the logs which are located in the directory *your_publish_root/var/log*.

DisplayDebugWarnings

Summary

Choose if debug warnings should be displayed explicitly on the top of the page or in the debug log only.

Usage

```
DisplayDebugWarnings=enabled|disabled
```

DebugRedirection

Summary

Enables debugging of internal and external module redirections.

Usage

```
DebugRedirection=enabled|disabled
```

Description

Whenever an internal redirection occurs the execution of eZ Publish will be stopped and a redirection page with a redirect button will appear. This allows you to see any errors that have occurred before the redirection takes place.

This setting is only useful for developers the eZ Publish core and custom extensions.

Debug

Summary

Sets if you want to display debug information in the rendered page or in a separate popup.

Usage

```
Debug=inline|popup
```

Description

Inline means that the debug output will be displayed together with the page that was rendered. Popup means that a separate popup window will be displayed with the debug information. Not all browsers support debug information in a popup. Choose inline if you have any problems.

Please note that if debug is set to "popup" in a virtual host environment then the following rewrite rules must be added to the virtual host block:

```
RewriteRule ^/var/cache/debug.html.* - [L]
RewriteRule ^/var/[^/]+/cache/debug.html.* - [L]
```

DebugIPList

Summary

Sets the hosts that receive debug output.

Usage

```
DebugIPList []
DebugIPList []= ip1|network1
DebugIPList []= ip2|network2
...
...
```

Description

The setting DebugByIP must be set to enabled to make this setting have any effect.

Examples

```
DebugIPList []=1.2.3.4
DebugIPList []=192.0.0.42
DebugIPList []=192.0.0.0/27
```

These settings enable debug output for the hosts 1.2.3.4, 192.0.0.42 and the network specified by the range 192.0.0.0/27

DebugByIP

Summary

Enables debug output for some IP addresses only. Useful when debugging live sites.

Usage

```
DebugByIP=enabled|disabled
```

Description

The setting DebugIPList controls which hosts will receive debug output. Debug output is sent to everyone if this setting is disabled.

ScriptDebugOutput

Summary

Enables debug output for PHP scripts run from the command line.

Usage

```
ScriptDebugOutput=enabled|disabled
```

Description

This setting relies on DebugOutput to be enabled.

DebugOutput

Summary

Main switch for debug output

Usage

```
DebugOutput=enabled|disabled
```

Description

Set this switch to enabled to turn on debug output in the rendered pages. There are several options that control what kind of debug output you will get.

[DesignSettings]

The DesignSettings are typically overridden for each siteaccess.

AdditionalSiteDesignList (page 1251)

Sets the additional site designs.

SiteDesign (page 1252)

Sets the most significant design resource.

StandardDesign (page 1253)

Sets the least significant (fallback) design resource.

AdditionalSiteDesignList

Summary

Sets the additional site designs.

Usage

```
AdditionalSiteDesignList []= design1  
AdditionalSiteDesignList []= design2  
...
```

Description

The design resources in AdditionalSiteDesignList[] are checked for templates after the sitedesign (page 1252) but before the standard design (page 1253). The designs are checked in the order they are listed.

You can read more about how design resources are used in the StandardDesign (page 1253) setting.

SiteDesign

Summary

Sets the most significant design resource.

Usage

```
SiteDesign= design_name
```

Description

This design resource is considered the most significant design resource. eZ publish will always search for templates in this design first.

You can read more about how design resources are used in the StandardDesign (page [1253](#)) setting.

Note: It is important that you specify all the designs used in the AvailableSiteDesigns setting in content.ini. You must remember to specify not only the sitedesigns used by this siteaccess, but all designs used by the complete site. F

StandardDesign

Summary

Sets the least significant (fallback) design resource.

Usage

```
StandardDesign= design_name
```

Description

The design resource set by StandardDesign is considered the least significant design resource. eZ publish will only use templates found in the standard design if the template could not be found in any of the other designs. It is generally a good idea not to change the standard design to anything different than *standard*. The design named *standard* comes with eZ publish and provides a default fallback template for all possible templates.

By default eZ publish will look for designs in the *design* directory in the eZ publish root. Using the setting DesignExtensions (page 1183) you can configure extensions to have designs as well.

For general information about designs and siteaccess read the SiteManagement (page 142) documentation.

The settings described here are usually overridden per siteaccess to provide a unique look.

Examples

```
StandardDesign=standard
SiteDesign=mydesign
AdditionalSiteDesignList []=base
AdditionalSiteDesignList []=extras
```

This setup will make eZ publish check the design directories in the following order:

1. mydesign
2. base
3. extras
4. standard

[ExtensionSettings]**ActiveAccessExtensions** (page 1255)

Sets the extensions that are available to eZ publish per siteaccess.

ActiveExtensions (page 1256)

Sets the extensions that are available to eZ publish.

ExtensionDirectory (page 1257)

Sets the directory where extensions are located.

ActiveAccessExtensions

Summary

Sets the extensions that are available to eZ publish per siteaccess.

Usage

```
ActiveAccessExtensions []= extension1
ActiveAccessExtensions []= extension2
```

Description

Each extension will have its settings automatically loaded. This setting works similarly to the ActiveExtensions setting but is loaded after the siteaccess is loaded. This means that the settings activated will only be available in the siteaccess this setting is specified.

ActiveExtensions

Summary

Sets the extensions that are available to eZ publish.

Usage

```
ActiveExtensions[] = extension1  
ActiveExtensions[] = extension2  
...
```

Description

Each extension will have its settings automatically loaded. This setting is loaded before the siteaccesses are loaded. Overriding this setting in a siteaccess has no effect.

Examples

```
ActiveExtensions[]  
ActiveExtensions[] = ezdhtml  
ActiveExtensions[] = myextension
```

This setup loads the extension ezdhtml (Online Editor) and your personal extension myextension.

ExtensionDirectory

Summary

Sets the directory where extensions are located.

Usage

```
ExtensionDirectory= path_relative_to_ez_publish_root
```

Description

This setting is set to extension by default. Do not change it unless you know what you are doing.

[FileSettings]**CacheDir** (page 1260)

Sets the directory where eZ publish stores cache files.

DirDepth (page 1262)

Sets the number of extra directories that will be made when storing a file.

LogDir (page 1259)

Sets the directory where eZ publish will store its logfiles.

StorageDir (page 1265)

Sets the directory eZ publish uses to store files.

StorageDirPermissions (page 1264)

Sets the permissions set on directories created in the storage directory.

StorageFilePermission (page 1263)

Sets the permissions set on files created in the storage directory.

TemporaryDir (page 1267)

Sets the directory eZ publish uses to store temporary files.

TemporaryPermissions (page 1266)

Sets the permissions on temporary files created by eZ publish

VarDir (page 1261)

Sets the main directory for file storage in eZ publish.

LogDir

Summary

Sets the directory where eZ publish will store its logfiles.

Usage

```
LogDir= directory_name
```

Description

The log files are stored within `/var_dir/ directory_name` where `var_dir` is specified by the `VarDir` (page 1261) setting.

Examples

```
VarDir=var  
LogDir=log
```

These settings will make eZ publish use the log dir ”/var/log”

CacheDir

Summary

Sets the directory where eZ publish stores cache files.

Usage

```
CacheDir= directory_name
```

Description

eZ publish creates many cache files to speed up execution. The cache files are stored within `/var_dir/ directory_name` where `var_dir` is specified by the `VarDir` (page 1261) setting.

Note: Don't change the default setting "cache" unless you know what you are doing.

Examples

```
VarDir=var  
CacheDir=cache
```

These settings will make eZ publish use the cache dir "/var/cache"

VarDir

Summary

Sets the main directory for file storage in eZ publish.

Usage

```
VarDir= directory_name
```

Description

The directory specified by vardir is used by eZ publish for file storage. This includes both content related files, cache files and other temporary files.

Note: Don't change the default setting "var" unless you know what you are doing.

Examples

```
VarDir=var
```

The directory "/var/" is used to store files.

DirDepth

Summary

Sets the number of extra directories that will be made when storing a file.

Usage

```
DirDepth= number
```

Description

Filesystems have limitations on how many files you can store within one directory. In order to avoid these problems eZ publish creates extra directories based on the first letters in the filename. This solution spreads the files over many directories. The DirDepth settings controls how many levels of extra directories eZ publish should make.

The default setting '3' support millions of files and should be enough for most sites.

Examples

```
DirDepth=3
```

If you store the file test.jpg eZ publish will store the file in the directory "t/e/s/" the complete path for the file will be "t/e/s/test.jpg".

StorageFilePermission

Summary

Sets the permissions set on files created in the storage directory.

Usage

```
StorageFilePermission= permission_setting
```

Description

Note that:

- It is important that the webserver has sufficient permissions to both write and remove directories.
- Setting the permissions to 0666 (read and write to all) is a potential security risk.

The preferred setting is 0660 (full read and write to user and group). This requires apache to have the correct user/group access.

StorageDirPermissions

Summary

Sets the permissions set on directories created in the storage directory.

Usage

```
StorageDirPermissions= permission_setting
```

Description

Note that:

- It is important that the webserver has sufficient permissions to both write and remove directories.
- Setting the permissions to 0777 (read and write to all) is a potential security risk.

The preferred setting is 0770 (full read and write to user and group). This requires apache to have the correct user/group access.

StorageDir

Summary

Sets the directory eZ publish uses to store files.

Usage

```
StorageDir= dir_name
```

Description

The directory name specified is relative to the var directory specified by VarDir (page 1261). The directory specified by StorageDir is used to store files related to the content of your site.

TemporaryPermissions

Summary

Sets the permissions on temporary files created by eZ publish

Usage

```
TemporaryPermissions= permission_setting
```

Description

The permission setting should be specified using the UNIX file permission schema.

Note that:

- It is important that the webserver has sufficient permissions to both write and remove files.
- Setting the permissions to 0777 (read and write to all) is a potential security risk.

The preferred setting is 0770 (full read and write to user and group). This requires apache to have the correct user/group access.

TemporaryDir

Summary

Sets the directory eZ publish uses to store temporary files.

Usage

```
TemporaryDir= dir_name
```

Description

eZ publish will use the directory name you choose inside `/ var_dir/ cache_dir/ dir_name` where `var_dir` is set by VarDir (page 1261) and `cache_dir` is set by CacheDir (page 1260).

This directory is only used to store files that are used during the rendering of one page. After the page is rendered any temporary files are removed.

[FormProcessSettings]**Module** (page 1269)

Sets if the form module should be enabled or not.

Module**Summary**

Sets if the form module should be enabled or not.

Usage

```
Module=enabled|disabled
```

Description

The form module is insecure by design and should not be used.

[InformationCollectionSettings]**EmailReceiver** (page 1271)

Sets the receiver of e-mail generated by the information collection system.

EmailReceiver**Summary**

Sets the receiver of e-mail generated by the information collection system.

Usage

```
EmailReceiver= e-mail_address
```

Description

Each time the information collector system receives data it is sent to the Email address specified by this setting.

[MailSettings]**AdminEmail** (page 1278)

Sets the mail address of the site administrator.

AllowedCharsets (page 1276)

Sets the character sets that eZ Publish sends directly in mail.

ContentType (page 1274)

Sets the content type for email sent from eZ Publish.

EmailSender (page 1277)

Sets the default sender address for mail sent from eZ Publish.

HeaderLineEnding (page 1273)

Sets the line ending character used in emails sent from eZ Publish.

OutputCharset (page 1275)

Sets the character set to convert mail into if they are formatted with the wrong character set.

Transport (page 1283)

Sets the mail transport system for mail sent from eZ Publish.

TransportPassword (page 1279)

Sets the password to use for authentication with the SMTP server.

TransportPort (page 1281)

Sets the port that should be used when connecting to the SMTP server.

TransportServer (page 1282)

Sets the hostname of the SMTP server.

TransportUser (page 1280)

Sets the user to use for authentication with the SMTP server.

HeaderLineEnding

Summary

Sets the line ending character used in emails sent from eZ Publish.

Usage

```
HeaderLineEnding=auto| url_encoded_value
```

Description

If you have problems with linebreaks in mail you may want to change this setting. Use URL a URL encoded value if you choose not to use the auto setting. E.g Specify Carriage Return with %0D and Line Feed with %0A.

Examples

```
HeaderLineEnding=%0A%0D
```

Sets linebreaks in e-mail to be LineFeed CarriageReturn.

ContentType**Summary**

Sets the content type for email sent from eZ Publish.

Usage

```
ContentType=text/plain|text/html
```

Description

The default setting is text/plain which means that mails sent are written in plain text. You can change this setting into text/html if you want to send HTML formatted mail. If you do this, you must also reformat all the mail templates with HTML tags.

OutputCharset

Summary

Sets the character set to convert mail into if they are formatted with the wrong character set.

Usage

```
OutputCharset= characterset
```

Description

If you try to send an e-mail formatted in character set not listed in AllowedCharsets (page [1276](#)) eZ Publish will automatically convert the mail into the character set specified.

AllowedCharsets

Summary

Sets the character sets that eZ Publish sends directly in mail.

Usage

```
AllowedCharsets[] = character set 1
AllowedCharsets[] = character set 2
...
...
```

Description

E-mail that are not in an accepted format will be converted to the format specified by OutputCharset (page [1275](#)).

Examples

```
AllowedCharsets[]
AllowedCharsets[] = us-ascii
AllowedCharsets[] = utf-8
```

With this setup eZ Publish will send all e-mail using the character sets us-ascii and utf-8 without conversion.

EmailSender

Summary

Sets the default sender address for mail sent from eZ Publish.

Usage

```
EmailSender= email_address
```

Description

The specified e-mail address will be used as the default value in the from field for mail sent from eZ Publish. If this field is left blank the value in AdminEmail (page 1278) is used instead.

AdminEmail**Summary**

Sets the mail address of the site administrator.

Usage

```
AdminEmail= email_address
```

Description

The admin email is used for notification mail for the administrator of the site. It is used throughout the system for important updates e.g when new users are created.

TransportPassword

Summary

Sets the password to use for authentication with the SMTP server.

Usage

```
TransportPassword= password
```

Description

You must set Transport (page 1283) to "smtp" for this setting to have any effect.

TransportUser**Summary**

Sets the user to use for authentication with the SMTP server.

Usage

```
TransportUser= username
```

Description

If your SMTP server requires authentication you must provide a username with this setting.

You must set Transport (page 1283) to "smtp" for this setting to have any effect.

TransportPort**Summary**

Sets the port that should be used when connecting to the SMTP server.

Description

You must set Transport (page 1283) to "smtp" for this setting to have any effect.

TransportServer

Summary

Sets the hostname of the SMTP server.

Usage

```
TransportServer= hostname
```

Description

You can use both a normal hostname and an IP address.

You must set Transport (page 1283) to "smtp" for this setting to have any effect.

Transport

Summary

Sets the mail transport system for mail sent from eZ Publish.

Usage

```
Transport=sendmail|smtp
```

Description

You can choose between sendmail and SMTP. The sendmail setting uses the sendmail client on the server. If you choose SMTP you must tell eZ Publish where your SMTP server is located using the TransportServer (page 1282), TransportPort (page 1281), TransportUser (page 1280) and TransportPassword (page 1279) settings.

[OverrideSettings]**Cache** (page 1285)

Sets if the template override cache should be enabled.

Cache**Summary**

Sets if the template override cache should be enabled.

Usage

```
Cache=enabled|disabled
```

Description

Note: Don't turn off the template override cache unless you know what you are doing.

[PortAccessSettings]**Portnumber to siteaccess mapping** (page 1287)

Creates a mapping between a portnumber and a siteaccess.

Portnumber to siteaccess mapping

Summary

Creates a mapping between a portnumber and a siteaccess.

Usage

```
number1= siteaccessname1  
number2= siteaccessname2
```

Description

You should have one line of this type for each port that is used to access your site.

Examples

```
80=user  
81=admin
```

This setup has two siteacceses. All requests to port 80 use the *user* siteaccess, while all requests to port 81 use the *admin* siteaccess.

[RegionalSettings]**ContentObjectLocale** (page 1296)

Sets the default language for content objects.

ContentXMLCharset (page 1295)

Sets the characterset used when storing XML in content objects.

Debug (page 1291)

Sets if debug mode should be enabled or disabled.

DevelopmentMode (page 1292)

Sets if development mode should be on or off.

HTTPLocale (page 1298)

Sets the locale transmitted to the web clients.

Locale (page 1299)

Sets the locale (currency, date and time settings etc.)

SystemLocale (page 1297)

Tells PHP to be run in a specific locale.

TextTranslation (page 1294)

Sets if text translation is enabled for template translation.

TranslationCache (page 1293)

Sets if the translation cache should be enabled or disabled.

TranslationExtensions (page 1289)

Sets the extensions that hold translations

TranslationRepository (page 1290)

Sets the default translation repository for eZ publish.

TranslationExtensions

Summary

Sets the extensions that hold translations

Usage

```
TranslationExtensions[] = extension1  
TranslationExtensions[] = extension2  
...
```

Description

eZ publish will search for additional translations in the "translations" directory of your extension. This setting is commonly overriden in the extension settings to tell eZ publish that the extension provides translations.

Examples

```
TranslationExtensions=myextension
```

eZ publish will now search the directory "/extension/myextension/translations" for additional translations.

TranslationRepository

Summary

Sets the default translation repository for eZ publish.

Usage

```
TranslationRepository= path_to_repository
```

Description

This setting should point to the default directory for eZ publish translations. This setting should be changed by developers only.

Debug**Summary**

Sets if debug mode should be enabled or disabled.

Usage

```
Debug=enabled|disabled
```

Description

If debug is enabled eZ publish will display information about the locale files that are loaded. You need to turn on Debug (page [1245](#)) globally to see debug output.

DevelopmentMode

Summary

Sets if development mode should be on or off.

Usage

```
DevelopmentMode=enabled|disabled
```

Description

Development mode makes eZ publish translate all untranslated string using bork mode. This can be handy when spotting untranslated text.

You should never use development mode in a production environment.

TranslationCache

Summary

Sets if the translation cache should be anabled or disabled.

Usage

```
TranslationCache=enabled|disabled
```

Description

This setting is for debugging purposes only. You should never use eZ publish without translation cache in a production environment.

TextTranslation

Summary

Sets if text translation is enabled for template translation.

Usage

```
TextTranslation=enabled|disabled
```

Description

This setting controls if eZ publish should translate strings marked with i18n in the templates. eZ publish runs slightly faster when this setting is set to "disabled". This setting is automatically set to disabled if your Locale (page 1299) is set to "eng-GB".

ContentXMLCharset

Summary

Sets the characterset used when storing XML in content objects.

Usage

```
ContentXMLCharset=enabled|disabled| character_set
```

Description

- **enabled** - Stores XML fields using the current character set. The current character set is set by the Charset (page 1191) setting. This setting is the correct setting for most people.
- **disabled** - Forces storage in UTF8.
- **character set** - Forces storage in this character set.

ContentObjectLocale

Summary

Sets the default language for content objects.

Usage

```
ContentObjectLocale= locale
```

Description

ContentObjectLocale defines the language that is considered the main language of your site. All content objects must exist in this language.

Note:Do not change this setting after you have run the setup wizard and added content to your site. Doing so will result in undefined behavior and data loss.

SystemLocale

Summary

Tells PHP to be run in a specific locale.

Usage

```
SystemLocale=locale_name1[,locale_name2][,locale_name3]...
```

Description

Use this directive to set locale for the entire PHP system (this functionality is similar to using the "["setlocale"](#)" PHP function where "LC_ALL" is passed as the first parameter). You can specify a comma separated list of locale names. If you use a character set specific locale (for example, "no_NO.UTF-8"), make sure this character set matches the output character set of eZ Publish (page [1191](#)). Please note that different operating systems have different naming schemes for locales, and thus you might need to use multiple names in order for your installation to work on multiple platforms (such as developing on Windows and running Linux in production).

Examples

Example 1

```
SystemLocale=de_DE.ISO-8859-1,german
```

This will tell PHP to first use the "de_DE.ISO-8859-1" locale, and if this is not available, then the "german" locale will be used.

Example 2

```
SystemLocale=no_NO.UTF-8,no_NO,norwegian
```

With this configuration each of the three listed elements (starting from "no_NO.UTF-8") will be tried to be set as new locale until success.

HTTPLocale

Summary

Sets the locale transmitted to the web clients.

Usage

```
HTTPLocale= locale_setting
```

Description

This setting is usually set automatically from the Locale (page 1299) setting. However you can override it for this setting. For most people the default setting (empty) is sufficient.

Locale

Summary

Sets the locale (currency, date and time settings etc.)

Usage

```
Locale= locale_name
```

Description

The locale controls settings related to country specific settings, e.g. language, currency and date and time formating. eZ publish provides many default locale settings in the "/share/locale" directory. Use the filename without the extension (.ini) to tell eZ publish to use that locale.

If you want custom locale settings, simply copy the locale configuration file that is the closest to the configuration you want. Then edit it and make it conform to your needs. Finally, tell eZ publish to use that locale using this setting.

Examples

```
Locale=nor-NO
```

Sets the locale to Norwegian settings.

[RoleSettings]**EnableCaching** ([page 1304](#))

Controls whether role caching should be enabled or disabled.

PolicyOmitList ([page 1302](#))

Excludes modules and views from the permission checking.

ShowAccessDeniedReason ([page 1301](#))

Sets if eZ Publish elaborates on the reason for getting access denied when viewing a page.

UserPolicyCache ([page 1303](#))

Sets which users' policies to cache.

ShowAccessDeniedReason

Summary

Sets if eZ Publish elaborates on the reason for getting access denied when viewing a page.

Usage

```
ShowAccessDeniedReason=enabled|disabled
```

Description

This option is for site debugging purposes and is disabled by default. Enabling elaborate messages can pose a security threat since details about the role setup of your system will be exposed.

PolicyOmitList

Summary

Excludes modules and views from the permission checking.

Usage

```
PolicyOmitList []= module1 [/ view1]
PolicyOmitList []= module2 [/ view2]
...
...
```

Description

This setting allows you to exclude complete modules or specific views from permission control. Modules and views excluded from permission control are always accessible for all users.

Examples

```
PolicyOmitList []=ezinfo
PolicyOmitList []=user/login
PolicyOmitList []=user/register
```

These settings excludes all the views in the ezinfo module, and the views login and register in the user module from permission checking.

UserPolicyCache

Summary

Sets which users' policies to cache.

Usage

```
UserPolicyCache=enabled|disabled| id1, id2, ...
```

Description

This setting has three possible values

1. **enabled** - Cache the policies of all users.
2. **disabled** - No policy cache will be stored.
3. **Comma separated list of IDs** - Store policy cache for all users with their IDs in this list.

Examples

```
UserPolicyCache=23,44,24
```

With this configuration eZ Publish will store policy cache for the users with IDs 23,24 or 44 only.

EnableCaching

Summary

Controls whether role caching should be enabled or disabled.

Usage

```
EnableCaching=true|false
```

Description

When role caching is enabled eZ Publish stores the complete permissions set for each user once it is computed. This saves both SQL queries and computation time on consequent requests.

Do not disable role caching unless you know what you are doing.

[RSSSettings]**AvailableVersionList** (page 1307)

Sets the available RSS versions

CacheTime (page 1308)

Sets the cachetime for RSS feeds in seconds.

DefaultVersion (page 1306)

Sets the default RSS version to use.

NumberOfObjectsDefault (page 1309)

Sets the default value for the number of items in an RSS export.

NumberOfObjectsList (page 1310)

Sets the number of items in an RSS export that the user can choose between in the administration interface.

DefaultVersion

Summary

Sets the default RSS version to use.

Usage

```
DefaultVersion= version
```

Description

This setting controls the RSS version that is selected by default when you create a new RSS export. The specified version must be found in the AvailableVersionList (page 1307) setting.

AvailableVersionList

Summary

Sets the available RSS versions

Usage

```
AvailableVersionList []= version1
AvailableVersionList []= version2
...
...
```

Description

This setting should only be changed by eZ Publish developers.

CacheTime**Summary**

Sets the cachetime for RSS feeds in seconds.

Usage

```
CacheTime= number_of_seconds
```

Description

In order to minimize the server load, generated RSS feeds are cached and updated at a regular interval. This setting controls the time between each update of the cache.

Changes to the content being fed will show up in the RSS feed after the number of seconds set at the latest.

NumberOfObjectsDefault

Summary

Sets the default value for the number of items in an RSS export.

Usage

```
NumberOfObjectsDefault= number
```

Description

The number must be present in the NumberOfObjectsList (page 1310) setting.

NumberOfObjectsList

Summary

Sets the number of items in an RSS export that the user can choose between in the administration interface.

Usage

```
NumberOfObjectsList []= number1  
NumberOfObjectsList []= number2  
...
```

Description

The administration interface displays a dropdown with the numbers set by this setting in the RSS export edit screen.. The dropdown is used to determine the number of items to export in the RSS feed.

[SearchSettings]**AllowEmptySearch** (page 1317)

Sets if users can search for nothing

DelayedIndexing (page 1312)

Sets if new content objects are indexed in the search engine upon publishing or if indexing is done by the cronjob

EnableWildcard (page 1316)

Sets if wildcard searching is allowed or not

LogSearchStats (page 1319)

Sets if search statistics should be saved or not.

MaximumSearchLimit (page 1318)

Sets the maximum number of returned hits.

MinCharacterWildcard (page 1315)

The minimum number of characters a wildcard can represent

SearchEngine (page 1321)

Sets which search engine to use

SearchViewHandling (page 1320)

Sets if searches are handled by the search view or in the template.

StopWordThresholdPercent (page 1313)

Sets the percentage of hits that a word should be present in before ignoring the word completely

StopWordThresholdValue (page 1314)

Sets the minimum number of objects in the database before the stopword functionality is used.

DelayedIndexing

Summary

Sets if new content objects are indexed in the search engine upon publishing or if indexing is done by the cronjob

Usage

```
DelayedIndexing=enabled|disabled
```

Description

Delaying the indexing means that some objects will not be found by the search engine even after they have been published. However, indexing can be CPU intensive and delaying the indexing can mean better performance and a more responsive administration interface for editors.

StopWordThresholdPercent

Summary

Sets the percentage of hits that a word should be present in before ignoring the word completely

Usage

```
StopWordThresholdPercent= number
```

Description

Searching for common words can lead to extremely many hits. In order to provide a better result the stop word system has been implemented. If a word is present in more than a set percentage of all the objects in the system then this word will be ignored when searched for.

Examples

```
StopWordThresholdPercent=60
```

If you search for a word that is present in more than 60% of all the objects then the search word will be ignored.

StopWordThresholdValue**Summary**

Sets the minimum number of objects in the database before the stopword functionality is used.

Usage

```
StopWordThresholdValue= number
```

Description

This setting should be used together with the StopWordThresholdPercent setting.

MinCharacterWildcard

Summary

The minimum number of characters a wildcard can represent

Usage

```
MinCharacterWildcard= number
```

Description

This setting controls the minimum number of characters required for the wildcard to match. If MinCharacterWildcard is set to 2 and you search for "dus**" then "dust" will not be found while "duster" will.

EnableWildcard**Summary**

Sets if wildcard searching is allowed or not

Usage

```
EnableWildcard=true|false
```

Description

When wildcard searching is enabled you can match on partial words. E.g searching for "dust*" will match both duster and dustdevil. Wildcard search requires a lot of resources and can heavily influence the performance of your site.

AllowEmptySearch

Summary

Sets if users can search for nothing

Usage

```
AllowEmptySearch=enabled|disabled
```

Description

Empty searches take a lot of resources and can slow down your site considerably. If you enable empty searches you must use template search view handling and make sure that your template submits enough limitations on the search.

MaximumSearchLimit

Summary

Sets the maximum number of returned hits.

Usage

```
MaximumSearchLimit= integer
```

Description

It is possible for the user to select the number of returned hits via a POST variable. In order to avoid very high limits resulting in high server load you can use this setting to set a cap on the number of returned hits.

Examples

```
MaximumSearchLimit=25
```

This setting will limit the maximum number of search hits to 25.

LogSearchStats

Summary

Sets if search statistics should be saved or not.

Usage

```
LogSearchStats=enabled|disabled
```

Description

If enabled statistics about each search are written to the database. You can view the statistics from the administration interface.

SearchViewHandling

Summary

Sets if searches are handled by the search view or in the template.

Usage

```
SearchViewHandling=default|template
```

Description

This setting has two options

- **default** - The view code does the search and passes the result to template. The template is only responsible for displaying the result.
- **template** - The template does the search and passes the result back to the view code. This way you have to "program" the search yourself providing more flexibility.

SearchEngine

Summary

Sets which search engine to use

Usage

```
SearchEngine= search_engine_identifier
```

Description

eZ Publish supports search engine plugins. This allows you to write support for your favorite search engine and to use this for the searching functionality in eZ Publish.

eZ Publish comes with built in support for two search engines:

- eZSearchEngine - The standard eZ Publish search engine
- openFts - The Open Source Full Text Search Engine

[Session]**ActivityTimeout** (page 1327)

Sets the number of second before a user is considered inactive.

CookieTimeout (page 1326)

Sets the number of seconds that the session cookie lasts.

SessionNameHandler (page 1325)

Sets how session names should be generated

SessionNamePerSiteAccess (page 1323)

Prepends session names with the current siteaccess.

SessionNamePrefix (page 1324)

Sets the prefix eZ publish should use when creating session names.

SessionTimeOut (page 1328)

Sets the number of seconds a session lasts.

SessionNamePerSiteAccess**Summary**

Prepends session names with the current siteaccess.

Usage

```
SessionNamePerSiteAccess=enabled|disabled
```

Description

When this setting is enabled the generated session names are prepended with the name of the siteaccess used to access the site. This generates unique session names per siteaccess.

SessionNamePrefix**Summary**

Sets the prefix eZ publish should use when creating session names.

Usage

```
SessionNamePrefix= prefix
```

Description

The prefix set in this variable is prepended to the generated session name. You can override this setting for each siteaccess to generate unique session names per siteaccess.

SessionNameHandler

Summary

Sets how session names should be generated

Usage

```
SessionNameHandler=default|custom
```

Description

There are two possible options

1. **default:** The default setting uses PHP to generate session names. If you use the default handler the session will be valid across siteaccess (e.g admin and user site) if they use the same database.
2. **custom:** The custom setting allows you to prefix the session name. The prefix is set with the SessionNamePrefix setting.

CookieTimeout**Summary**

Sets the number of seconds that the session cookie lasts.

Usage

```
CookieTimeout= number
```

Description

This setting is set in the cookie used by the client browsers to store the login information. When the cookie times out it will be removed by the client browser. Note that the maximum session duration is the minimum of this setting and the SessionTimeout setting.

ActivityTimeout

Summary

Sets the number of second before a user is considered inactive.

Usage

```
ActivityTimeout= number
```

Description

This setting is only used for lists and statistical purposes in the setup area in the administration interface. It does not have any impact on the session duration.

SessionTimeOut**Summary**

Sets the number of seconds a session lasts.

Usage

```
SessionTimeOut= number
```

Description

After the specified time has passed the session will no longer be considered valid, and the user will be logged out even if he/she is active on the site.

[SetupSettings]**CriticalTests** (page 1332)

Sets the tests that must be passed in order to complete the setup procedure.

OptionalTests (page 1331)

Sets the tests optional tests run during the setup procedure.

OverrideSiteDesign (page 1330)

Sets the sitedesign that should be used by the setup wizard.

PageLayout (page 1333)

Sets which pagelayout template to use for the setup wizzard.

OverrideSiteDesign

Summary

Sets the sitedesign that should be used by the setup wizard.

Usage

```
OverrideSiteDesign= sitedesign_name
```

OptionalTests**Summary**

Sets the tests optional tests run during the setup procedure.

Usage

```
OptionalTests= test1, test2,...
```

Description

Optional tests can be bypassed simply by clicking next.

CriticalTests**Summary**

Sets the tests that must be passed in order to complete the setup procedure.

Usage

```
CriticalTests= test1, test2,...
```

PageLayout

Summary

Sets which pagelayout template to use for the setup wizzard.

Usage

```
PageLayout= template_name
```

Description

Don't change this setting unless you are an eZ publish developer.

[ShopSettings]**ClearBasketOnCheckout** (page 1336)

Sets when the basket is cleared.

RedirectAfterAddToBasket (page 1335)

Controls where the user is redirected after adding an item to the basket.

RedirectAfterAddToBasket

Summary

Controls where the user is redirected after adding an item to the basket.

Usage

```
RedirectAfterAddToBasket=basket|reload
```

Description

This setting has two possible values:

1. *basket* - Redirect the user back to the basket to show the newly added item.
2. *reload* - Redirect the user back to the page he came from, this allows the user to continue shopping.

ClearBasketOnCheckout

Summary

Sets when the basket is cleared.

Usage

```
ClearBasketOnCheckout=enabled|disabled
```

Description

This setting has two possible values:

1. *disabled* - Means that the basket is cleared when the shop/checkout trigger is done. In practice this means when a user has payed the product and payment system is finished. This is the default value since it means the user can cancel the order and go back to the shop with the basket still intact.
2. *enabled* - Means to clear the basket as soon as the user clicks confirm in the shop/confirmorder trigger. This may needed by some payment systems. Check the documentation for your payment system to see if this setting must be enabled. The impact on the users is that the basket will not be available when the payment is cancelled.

[SiteAccessRules]**Rules** (page 1338)

Sets which modules and views to enable or disable.

Rules

Summary

Sets which modules and views to enable or disable.

Usage

```
Rules[] =function1;parameter1  
Rules[] =function2;parameter2
```

Description

The "Rules" setting defines a ruleset that eZ publish uses to determine whether module and views should be available or not. Each line of the ruleset is read and evaluated. The order in which you enable/disable modules does not matter, but more specific rules will override global rules.

The available functions are:

access

The access function is used to change if subsequent *module* or *moduleall* rules should be enabled or disabled. The access function has two possible parameter options

1. **enable** - enable subsequently mentioned modules and views
2. **disable** - disables subsequently mentioned modules and views

module

The module function is used to specify complete modules or specific views.

The module or view will get the access level of the last access level set. To specify a module simply provide the module name as the parameter value:

```
Rules[] =module;name_of_module
```

If you specify a module, the access mode is set for all views in that module.

To specify a single view you need to specify both the name of the module and the view as the parameter value:

```
Rules[] =module;name_of_module/name_of_view
```

moduleall

Moduleall is a special mode that should be used on a line on itself without a trailing function definition. It is used to set the current access mode to all modules in the system.

Examples

Example 1

```
Rules []=access;enable  
Rules []=moduleall
```

This configuration enables all modules and views. This is equal to the default ruleset.

Example 2

```
Rules []=access;enable  
Rules []=moduleall  
Rules []=access;disable  
Rules []=module;ezinfo  
Rules []=module;content/search
```

This configuration enables all modules and views except all views in the ezinfo module and the view content/search.

Example 3

```
Rules []=access;disable  
Rules []=moduleall  
Rules []=access;enable  
Rules []=module;content/view
```

This configuration takes a different approach. It disables all modules by default and enables only the modules that you want to use. In this case only the content/view module is available.

[SiteAccessSettings]**AnonymousAccessList** (page 1359)

Specifies a lists of modules and views that are accessible regardless of the RequireUserLogin setting.

AvailableSiteAccessList (page 1362)

Sets the siteaccesses that your eZ Publish installation provides.

CheckValidity (page 1364)

Sets if the setup wizard should be activated or not.

DebugAccess (page 1361)

Turns on debug output for access matching.

DebugExtraAccess (page 1360)

Enables verbose access matching debug information.

ForceVirtualHost (page 1365)

Sets if eZ Publish should force virtual host mode.

HostMatchElement (page 1350)

Sets which element (separated by a dot) to use when using element host matching.

HostMatchMapItems (page 1345)

Sets the mapping between hostname and siteaccess when using map host-matching.

HostMatchRegexp (page 1349)

Sets the regular expression that is used to fetch the siteaccess from the hostname with regexp hostmatching.

HostMatchRegexpItem (page 1348)

Sets which submatch to use to determine the siteaccess name with regexp hostmatching.

HostMatchSubtextPost (page 1346)

Sets the postfix that occurs in the hostname behind the siteaccess when using text host matching.

HostMatchSubtextPre (page 1347)

Sets the prefix that occurs in the hostname in front of the siteaccess when using text host matching.

HostMatchType (page 1351)

Sets which type of host matching to use to select the siteaccess.

MatchOrder (page 1358)

Sets the matching algorithms that will be tried to determine which siteaccess to use.

PathPrefix (page 1344)

Not documented yet.

RequireUserLogin (page 1363)

Sets if you allow anonymous access to your site.

ServerVariableName (page 1343)

Not documented yet.

ShowHiddenNodes (page 1342)

Sets if hidden nodes should be shown by default or not.

URIMatchElement (page 1355)

Specifies which element (separated by "/") of the URI that contains the siteaccess.

URIMatchRegexp (page 1354)

Sets the regular expression that is used to extract the siteaccess from the URI.

URIMatchRegexpItem (page 1353)

Sets the submatch that is used for siteaccess matching when using regexp URIMatching.

URIMatchType (page 1356)

Configures the URI match access method

ShowHiddenNodes

Summary

Sets if hidden nodes should be shown by default or not.

Usage

```
ShowHiddenNodes=true|false
```

Description

eZ Publish has the concept of node visibility (page [132](#)). These settings controls whether hidden nodes should be displayed by default or not. The settings controls the behavior of the content fetch functions. ShowHiddenNodes is typically overridden for siteaccesses where all content should be shown regardless of the state, e.g in the administration interface.

ServerVariableName**Summary**

Not documented yet.

Usage

ServerVariableName

PathPrefix**Summary**

Not documented yet.

Usage

```
PathPrefix= prefix
```

HostMatchMapItems

Summary

Sets the mapping between hostname and siteaccess when using map host-matching.

Usage

```
HostMatchMapItems[] = hostname1; siteaccess1  
HostMatchMapItems[] = hostname2; siteaccess2  
...
```

Description

Each row in the array defines one hostname to match and the siteaccess that will be used for that hostname.

See HostMatchType (page [1351](#)) for examples and more information about host matching.

HostMatchSubtextPost

Summary

Sets the postfix that occurs in the hostname behind the siteaccess when using text host matching.

Usage

```
HostMatchSubtextPost= number
```

Description

Use this setting together with HostMatchSubtextPre (page 1347) to configure text host-matching.

See HostMatchType (page 1351) for examples and more information about host matching.

HostMatchSubtextPre

Summary

Sets the prefix that occurs in the hostname in front of the siteaccess when using text host matching.

Usage

```
HostMatchSubtextPre= text
```

Description

Use this setting together with HostMatchSubtextPost to configure text host-matching.
See HostMatchType (page 1351) for examples and more information about host matching.

HostMatchRegexpItem

Summary

Sets which submatch to use to determine the siteaccess name with regexp hostmatching.

Usage

```
HostMatchRegexpItem= number
```

Description

The setting HostMatchRegexp (page 1349) sets the regular expression that is used for matching.

See HostMatchType (page 1351) for examples and more information about host matching.

HostMatchRegexp

Summary

Sets the regular expression that is used to fetch the siteaccess from the hostname with regexp hostmatching.

Usage

```
HostMatchRegexp= regular_expression
```

Description

The regular expression should contain at least one submatch. The setting HostMatchRegexpItem (page 1348) controls which submatch that is used for the siteaccess name.

See HostMatchType (page 1351) for examples and more information about host matching.

HostMatchElement**Summary**

Sets which element (separated by a dot) to use when using element host matching.

Usage

```
HostMatchElement= number
```

Description

See HostMatchType (page 1351) for examples and more information about host matching.

HostMatchType

Summary

Sets which type of host matching to use to select the siteaccess.

Usage

```
HostMatchType=disabled|map|element|text|regexp
```

Description

Use host match when you want to select the siteaccess based on the host part of the URL. Host matching can be used in four different modes or be disabled completely. The available options are:

- **disabled** - Disables host matching. If host matching is specified in the MatchOrder (page 1358) setting it will be ignored.
- **map** - Select the siteaccess based on the complete hostname. Specify the mapping between a hostname and the siteaccess to use with the HostMatchMapItems (page 1345) setting.
- **element** - The siteaccess is specified by a one of the subdomains that are part of the hostname. Select which part of the hostname (separated by dots) that specifies the siteaccess with the HostMatchElement (page 1350) setting.
- **text** - The siteaccess is given by some arbitrary part of the hostname with a fixed pre and post text. Set the pre and post text with the HostMatchSubtextPre (page 1347) and HostMatchSubtextPost (page 1346) settings.
- **regexp** - Use a regular expression to obtain the siteaccess name from the hostname. Set the regular expression and the submatch to use with the HostMatchRegexp (page 1349) and HostMatchRegexpItem (page 1348) settings.

Examples

Map matching

```
HostMatchType=map
HostMatchMapItems []=mydomain.no;user
HostMatchMapItems []=admin.mydomain.no;admin
```

This configuration uses the map hostmatching type. If the site is accessed using the domain mydomain.no the siteaccess named *user* is used. If the domain admin.mydomain.no is used the *admin* siteaccess is used instead.

Element matching

```
HostMatchType=element  
HostMatchElement=1
```

This configuration uses the second element in the hostname as siteaccess using element matching. For example the hostname first.second.third.com yields the siteaccess *second*.

Text matching

```
HostMatchType=text  
HostMatchSubtextPre=my  
HostMatchSubtextPost=site.ez.no
```

This text matching configuration uses everything between my and site.ez.no as the siteaccess. E.g myadminsite.ez.no would produce the siteaccess *admin*. The hostname myusersite.ez.no produces the siteaccessname *user*.

Regexp matching

```
HostMatchType=regexp  
HostMatchRegexp=^(.+)\.example\.com  
HostMatchRegexpItem=1
```

This regexp setup uses the regular expression `^(.+)\.example\.com` to match anything in front of .example.com. The first match is set up to hold the siteaccess. E.g the hostname test.example.com yields the siteaccess *test*.

URIMatchRegexpItem

Summary

Sets the submatch that is used for siteaccess matching when using regexp URIMatching.

Usage

```
URIMatchRegexpItem= number
```

Description

The setting URIMatchRegexp (page 1354) specifies the regular expression that should be used for matching.

See URIMatchType (page 1356) for examples and more information about URI matching.

URIMatchRegexp

Summary

Sets the regular expression that is used to extract the siteaccess from the URI.

Usage

```
URIMatchRegexp= regular_expression
```

Description

Use the setting URIMatchRegexpItem (page 1353) to determine the submatch that contains the siteaccess.

See URIMatchType (page 1356) for examples and more information about URI matching.

URIMatchElement

Summary

Specifies which element (separated by "/") of the URI that contains the siteaccess.

Usage

```
URIMatchElement= number
```

Description

See URIMatchType (page 1356) for examples and more information about URI matching.

URIMatchType

Summary

Configures the URI match access method

Usage

```
URIMatchType=disabled|map|element|text|regexp
```

Description

URI matching is used to select the siteaccess from the location part of the URL: URL matching can be used in two different modes or be disabled completely. The options you have for this settings are:

- **disabled** - Disables URI matching. If URI matching is present in the MatchOrder (page 1358) setting it will be ignored.
- **map** - Select the siteaccess to use based on the first element of a given URL. Specify the mapping between the URL element and the siteaccess to use with the URIMatchMapItems setting.
- **element** - Select the siteaccess to use based on the elements separated by "/" in the location part. If you specify "URIMatchElement=N", the system will take N parts of the URL (separated by "/") and replace slashes with underscores in the resulting string.
- **text** - Match URL using pre- or post- substrings (URIMatchSubtextPre, URIMatchSubtextPost).
- **regexp** - Works similarly to the element setting but you can specify the delimiter with a regular expression. The settings URIMatchRegExp and URIMatchRegExpItem control how to split the URL and what part to use for the siteaccess.

Examples

Example 1 (element matching)

```
URIMatchType=element
URIMatchElement=1
```

With this configuration the second element (the first element is element 0) of the URI will be used as the siteaccess. For example, in the URL "http://www.mysite.com/news/site/admin/content/view/full/32" the siteaccess name will be "news".

Example 2 (element matching)

```
URIMatchType=element  
URIMatchElement=3
```

With this configuration the second element (the first element is element 0) of the URI will be used as the siteaccess. For example, in the URL "http://www.mysite.com/news/site/admin/content/view/full/32" the siteaccess name will be "news_site_admin".

Example 3 (map matching)

```
URIMatchType=map  
URIMatchMapItems []=myadmin_de;admin_de  
URIMatchMapItems []=MYADMIN_DE;admin_de
```

This will tell the system to use the "admin_de" siteaccess when the first element of a given URI is 'myadmin_de' or 'MYADMIN_DE'.

MatchOrder

Summary

Sets the matching algorithms that will be tried to determine which siteaccess to use.

Usage

```
MatchOrder= match_type1 [; match_type2] ...
```

Description

You can specify several matching types separated by a semicolon. eZ Publish will try each matching type, in the order specified, until a match is found.

The possible algorithms are:

- **host** - Fetch the siteaccess from the hostname. E.g admin.mydomain.com for the admin site and www.mydomain.com for the user site. Use the settings starting with HostMatch to configure host matching.
- **uri** - Fetch the siteaccess from the requested path. E.g mydomain.com/index.php/*siteaccess_name*/ . If you omit the siteaccess in the URL the default siteaccess will be used. Use the settings starting with UriMatch to configure URI matching.
- **port** - Select the siteaccess based on the port used to access the site. E.g mydomain.com for the user site, and mydomain.com:99 for the administration interface. Use the [PortAccessSettings] group to configure port matching.
- **servervar** - Fetches the siteaccess to use from a server variable found in the PHP \$_SERVER global. Use the ServerVariableName setting to choose the server variable that contains the siteaccess to use.

The most common access methods are described in detail in the access methods (page 146) chapter.

AnonymousAccessList

Summary

Specifies a lists of modules and views that are accessible regardless of the RequireUserLogin setting.

Usage

```
AnonymousAccessList []= module1/ view1  
AnonymousAccessList []= module2/ view2  
...
```

Description

Using the RequireUserLogin (page 1363) setting you can force users to log in in order to get access to a site. However, those users still need access to the log in page in order to log in. This setting controls the modules and views that are accessible regardless of the RequireUserLogin (page 1363) setting.

The default settings allow access to the modules and views related to logging in and registering a user.

Examples

```
AnonymousAccessList []=user/register  
AnonymousAccessList []=user/forgotpassword
```

These settings allow access to the register page and the forgotpassword page even if RequireUserLogin is enabled.

DebugExtraAccess

Summary

Enables verbose access matching debug information.

Usage

```
DebugExtraAccess=enabled|disabled
```

Description

This option is for eZ Publish developers only.

DebugAccess

Summary

Turns on debug output for access matching.

Usage

```
DebugAccess=enabled|disabled
```

Description

This option is for eZ Publish developers only.

AvailableSiteAccessList

Summary

Sets the siteaccesses that your eZ Publish installation provides.

Usage

```
AvailableSiteAccessList []=site_access_name1  
AvailableSiteAccessList []=site_access_name2  
...
```

Description

This setting controls the siteaccesses that are available from eZ Publish. Most installations have two siteaccesses: the user site and the administration interface.

This setting has to be set in settings/override/site.ini.append(.php). eZ Publish will use the matching rules to decide which siteaccess a user wants to access. More information about the site access system can be found in the site management (page [142](#)) chapter.

Examples

```
AvailableSiteAccessList []=example  
AvailableSiteAccessList []=example_admin
```

This system has two siteaccesses: example and example_admin. Specific override settings for the public siteaccess can be found in the directory settings/siteaccess/example/. Specific override settings for the admin siteaccess can be found in the directory settings/siteaccess/example_admin/.

RequireUserLogin

Summary

Sets if you allow anonymous access to your site.

Usage

```
RequireUserLogin=true|false
```

Description

If this setting is set to true, eZ Publish will redirect all requests from users that are not logged in to the log in page.

This setting is typically overridden for each siteaccess. Private siteaccesses like intranets or the admin interface will typically have RequireUserLogin set to true while public sites that everyone can access typically have RequireUserLogin set to false.

CheckValidity**Summary**

Sets if the setup wizard should be activated or not.

Usage

```
CheckValidity=true|false
```

ForceVirtualHost

Summary

Sets if eZ Publish should force virtual host mode.

Usage

```
ForceVirtualHost=true|false
```

Description

Normally eZ Publish automatically detects if virtual host or non virtualhost should be used. However, some special cases require virtual host mode to be forced.

You can read more about the various access methods for eZ Publish in the Access Methods (page [146](#)) chapter. Specific information on virtual host setups can be found in the Virtual host setup (page [70](#)) chapter.

[SiteSettings]**DefaultAccess** (page 1370)

Sets the default site access when the URI access method is used.

IndexPage (page 1371)

Sets the page to display when the root "/" of your site is accessed.

LoginPage (page 1369)

Sets if eZ Publish should use a custom pagelayout for the log in page.

MetaDataArray (page 1372)

Sets the site metadata that is used on several places on your site.

SiteList (page 1367)

Sets the siteaccesses available to outside sources (currently used for webdav only)

SiteName (page 1374)

The name of your site.

SiteURL (page 1373)

Sets the URL of your site. Used e.g when generating links for notifications and emails.

SSLPort (page 1368)

The port that should be used for SSL requests.

SiteList**Summary**

Sets the siteaccesses available to outside sources (currently used for webdav only)

Usage

```
SiteList []= siteaccess1  
SiteList []= siteaccess2  
...
```

SSLPot**Summary**

The port that that should be used for SSL requests.

Usage

```
SSLPot= integer
```

Description

Note: The default 443 should suffice for most users.

LoginPage

Summary

Sets if eZ Publish should use a custom pagelayout for the log in page.

Usage

```
LoginPage=custom|embedded
```

Description

If LoginPage is set to custom, eZ Publish will use loginpagelayout.tpl when users access the login page.

If LoginPage is set to embedded the normal pagelayout.tpl will be used.

DefaultAccess

Summary

Sets the default site access when the URI access method is used.

Usage

```
DefaultAccess= siteaccess_name
```

Description

When you use the URI access method (page 146) it is not possible to see which siteaccess you want to use if you are accessing the root of your site. The DefaultAccess setting specifies which siteaccess to choose when this happens.

IndexPage

Summary

Sets the page to display when the root "/" of your site is accessed.

Usage

```
IndexPage= internal url
```

Description

Use either a system or a nice URL to specify the index page. System URL's are a bit faster and should be used for the IndexPage setting since the root of your site will be accessed very often.

Examples

```
IndexPage=/content/view/full/2
```

This example shows the full view of node 2 as the index page of your site.

MetadataArray

Summary

Sets the site metadata that is used on several places on your site.

Usage

```
MetadataArray[ meta_name1]= meta_data1  
MetadataArray[ meta_name2]= meta_data2  
...
```

Description

This setting can be used to control meta data that is used in several different locations in your site. The default installation defines author, copyright, description and keywords. You are free to define your own metadata. To fetch the metadata use the ezini template operator.

Examples

```
MetadataArray[author]=eZ Systems  
MetadataArray[copyright]=eZ Systems  
MetadataArray[description]=Content Management System  
MetadataArray[keywords]=cms, publish, e-commerce, content management, development framework
```

SiteURL**Summary**

Sets the URL of your site. Used e.g when generating links for notifications and emails.

Usage

```
SiteURL= site_url
```

Description

The URL should contain the full hostname and any additional path to the root of your eZ Publish installation.

Examples

```
SiteURL=ez.no
```

A typical generated link will now look like `http://ez.no/some_url_here`.

SiteName**Summary**

The name of your site.

Usage

```
SiteName=the name of your site
```

Description

This setting is used in the title of the default templates. You may use spaces in the title.

Examples

```
SiteName=Sigges megaphone warehouse
```

[TemplateSettings]

You can read more about the template system in the templates (page 165) chapter.

AutoloadPathList (page 1387)

Sets the directories inside eZ publish itself where eZ publish will look for operator and function definitions.

Debug (page 1385)

Turns on/off template debug output.

ExtensionAutoloadPath (page 1386)

Sets the extensions that contain template function or operator definitions.

NodeTreeCaching (page 1381)

Sets if the template interprator should cache parsed template files.

ShowMethodDebug (page 1383)

Sets if debug information about called functions and operators should be displayed.

ShowUsedTemplates (page 1382)

Enables a table in the debug displaying all the templates used to render the current page.

ShowXHTMLCode (page 1384)

Sets if eZ publish should display template load debug inline or not.

TemplateCache (page 1377)

Main switch for all the template related caches.

TemplateCompile (page 1380)

Sets if the template compiler should be used or not.

TemplateCompression (page 1376)

Sets if compiled templates should be compressed or not.

TemplateOptimization (page 1379)

Sets if the template compiler should try to optimize the produced PHP code.

UseFormatting (page 1378)

Sets if the template compiler should keep whitespace in the compiled template.

TemplateCompression

Summary

Sets if compiled templates should be compressed or not.

Usage

```
TemplateCompression=enabled|disabled
```

Description

Enable template compression to make eZ publish store compiled templates using gzip. This saves a lot of space but is a bit slower.

TemplateCache

Summary

Main switch for all the template related caches.

Usage

```
TemplateCache=enabled|disabled
```

Description

If you disable TemplateCache the template system will not do any caching at all. This includes cache-blocks and compiled templates.

Note: Template caching is essential to speed up your site. Do not turn TemplateCache off unless you are developing.

UseFormatting

Summary

Sets if the template compiler should keep whitespace in the compiled template.

Usage

```
UseFormatting=enabled|disabled
```

Description

Enabling this setting makes the template compiler keep whitespace formatting in the compiled templates. This results in HTML output that bigger but easier to read.

This setting has no effect if TemplateCompile (page 1380) is disabled.

TemplateOptimization

Summary

Sets if the template compiler should try to optimize the produced PHP code.

Usage

```
TemplateOptimization=enabled|disabled
```

Description

Optimized templates will be slightly faster than templates that are not compiled.

This setting has no effect if TemplateCompile (page 1380) is disabled.

TemplateCompile

Summary

Sets if the template compiler should be used or not.

Usage

```
TemplateCompile=enabled|disabled
```

Description

If you enable TemplateCompile eZ publish will convert your templates into executable PHP files. If this option is disabled eZ publish will interpret each template separately for each page.

eZ publish will compile the templates on demand. Once a template is compiled it takes less time to process. However, compiling templates takes a lot of time. This can be noticed after you have cleared the cache. If this is a problem for your site you can use the "bin/php/eztc.php" script to compile all your templates before you go live.

NodeTreeCaching

Summary

Sets if the template interpreter should cache parsed template files.

Usage

```
NodeTreeCaching=enabled|disabled
```

Description

Enable this setting to make eZ publish go faster if you have disabled TemplateCompile (page 1380). If TemplateCompile (page 1380) is enabled this setting will have no effect.

ShowUsedTemplates

Summary

Enables a table in the debug displaying all the templates used to render the current page.

Usage

```
ShowUsedTemplates=enabled|disabled
```

Description

Debug (page 1385) must be enabled for this setting to have any effect.

Note: If you have caching enabled you will have to clear the cache for the table to display all templates used to render a page.

ShowMethodDebug

Summary

Sets if debug information about called functions and operators should be displayed.

Usage

```
ShowMethodDebug=enabled|disabled
```

Description

Debug (page 1385) must be enabled for this setting to have any effect.

The extra debug output provided by this setting is only useful for kernel developers.

ShowXHTMLCode

Summary

Sets if eZ publish should display template load debug inline or not.

Usage

```
ShowXHTMLCode=enabled|disabled
```

Description

If ShowXHTMLCode is enabled eZ publish will display a comment in the rendered output of the browser for each time a new template is loaded.

Debug (page [1385](#)) must be enabled for this setting to have any effect.

Note: If you have caching enabled you may will have to clear the cache for this setting to take effect.

Debug

Summary

Turns on/off template debug output.

Usage

```
Debug=enabled|disabled
```

Description

If you enable this setting eZ publish will display a list of debug messages at the bottom of each page. The output is configurable but will typically consist of information about the time used to render the page, warnings or errors while executing your template, information about the SQL queries executed and a list of the templates used.

Turning on debug will also make eZ publish output comments in the HTML output each time it loads a new template.

If things don't work as you expected while developing your site, turn on Debug and make sure that there are no warnings or errors while rendering your page.

Note: If you have caching enabled, some debug messages may disappear the second time you load a page. This is because the cache was used instead of rerendering the page. If you want to see all the debug messages you have to clear the cache or turn off caching altogether.

ExtensionAutoloadPath

Summary

Sets the extensions that contain template function or operator definitions.

Usage

```
ExtensionAutoloadPath[] = directory_name1
ExtensionAutoloadPath[] = director_namey2
...
...
```

Description

eZ publish will look for the file eztemplateautoload.php inside the directory ”/extension/*directory_name*/ autoloads”. The eztemplateautoload.php file should contain the function and operator definitions provided by the corresponding extension.

This setting is typically set in the settings for each extension that contain template operators or functions.

Examples

```
ExtensionAutoloadPath[] =myextension
```

This setting will make eZ publish load template function and operator definitions from the file ”/extension/myextension/autoloads/eztemplateautoload.php”

AutoloadPathList

Summary

Sets the directories inside eZ publish itself where eZ publish will look for operator and function definitions.

Usage

```
AutoloadPathList[] = directory1
AutoloadPathList[] = directory2
...
...
```

Description

If you want to specify additional paths that eZ publish should search for template operators and functions use the ExtensionAutoLoadPath setting.

Note: Do not change this setting unless you know what you are doing.

[TipAFriend]**FromEmail** (page 1389)

Sets the from address used when sending tip a friend e-mail.

MaxRequestsPerTimeframe (page 1390)

Set the maximum number of tip a friend mail the system will send to one email address.

TimeFrame (page 1391)

Sets the time frame for the tip a friend functionality.

FromEmail**Summary**

Sets the from address used when sending tip a friend e-mail.

Usage

```
FromEmail= mail_address
```

MaxRequestsPerTimeframe

Summary

Set the maximum number of tip a friend mail the system will send to one email address.

Usage

```
MaxRequestsPerTimeframe= number
```

Description

eZ Publish will not allow anyone to send more than MaxRequestsPerTimeframe tip a friend mail per TimeFrame (page 1391) to any single mail address. This prevents the possibility to spam a random email addresses.

Examples

```
MaxRequestsPerTimeframe=10  
TimeFrame=2
```

Using these settings eZ Publish will not allow users to send more than 10 tip a friend mails to a single email address per two hours.

TimeFrame**Summary**

Sets the time frame for the tip a friend functionality.

Usage

```
TimeFrame= number_of_hours
```

Description

You can control the maximum number of tip a friend mail the system can send to one address using this setting and the MaxRequestsPerTimeframe (page 1390) setting.

[UnitSettings]**BinaryUnits** (page 1394)

Sets which units eZ publish considers binary units.

UseSIUnits (page 1393)

Sets if eZ publish should use OSI or SI prefixes for binary numbers

UseSIUnits

Summary

Sets if eZ publish should use OSI or SI prefixes for binary numbers

Usage

```
UseSIUnits=true|false
```

Description

If this setting is set to true eZ publish will use SI prefixes for binary numbers. If the setting is set to false eZ publish uses OSI prefixes for binary numbers. You can set the units that eZ publish should consider binary with the BinaryUnits (page [1394](#)) setting.

BinaryUnits

Summary

Sets which units eZ publish considers binary units.

Usage

```
BinaryUnits= unit1; unit2;...
```

Description

Binary units will get binary prefixes in front of the 'B' for byte or 'b' for bit. The possible units are listed in units.ini (page [1434](#)).

Use the UseSIUnits (page [1393](#)) settings to force eZ publish to use SI prefixes instead of the standard OSI approved prefixes.

The configuration file units.ini (page [1434](#)) allows you to alter the prefixes.

Examples

OSI prefixes

The binary number 1 048 576 byte will be displayed as 1 MB

The binary number 1 048 576 bit will be displayed as 1 Mb

SI prefixes

The binary number 1 048 576 byte will be displayed as 1 MiB

The binary number 1 048 576 bit will be displayed as 1 Mib

[URLTranslator]**MaximumWildcardIterations** (page 1396)

Sets how many times the wildcard matches can iterate

Translation (page 1398)

Enables/disables the URL translation functionality

WildcardTranslation (page 1397)

Enables subtree URL translation

MaximumWildcardIterations

Summary

Sets how many times the wildcard matches can iterate

Usage

```
MaximumWildcardIterations= number_of_iterations
```

Description

The wildcard matcher system will iterate if the translated url is not a complete url, this allows urls lookup to be recursive and is required for proper subtree history.

Note:This is an implementation specific setting which should not be changed unless you know what you are doing.

WildcardTranslation

Summary

Enables subtree URL translation

Usage

```
WildcardTranslation=enabled|disabled
```

Description

Wildcard translations can be used to translate a complete node tree into another one. This is used for example if you move a complete tree of nodes. It is also possible to specify wildcard translations from the admin interface.

Note: Do not disable this option unless you are having problems with wildcardtranslations.

Translation

Summary

Enables/disables the URL translation functionality

Usage

```
Translation=enabled|disabled
```

Description

When URL translation is enabled, eZ Publish automatically generates easy URL's for all content objects based on the object names. These URL's can be used instead of using system URL's. Additionally the custom URL translation system (available from the admin interface) is enabled.

More information about URL translation can be found in the URL translation (page [152](#)) section

[UserSettings]**AnonymousUserID** (page 1418)

Sets the eZ Publish user that should be used for anonymous page requests.

AuthenticateMatch (page 1406)

Sets the fields that are accepted for login authentication.

DefaultSectionID (page 1416)

Sets the section ID of self registered users.

DefaultUserPlacement (page 1417)

Sets the parent object of all users who register themselves

ExtensionDirectory (page 1401)

Sets the extensions that contain login handlers

GeneratePasswordIfEmpty (page 1420)

Sets if eZ Publish should generate passwords if the password field is empty.

GeneratePasswordLength (page 1419)

Sets the length of passwords generated by eZ Publish.

HashType (page 1408)

Sets the algorithm used to encrypt the user passwords stored in the database.

LoginHandler (page 1402)

Sets the various methods eZ Publish will try to authenticate user logins.

LogoutRedirect (page 1403)

Sets which page to redirect to when a user has logged out.

RegistrationEmail (page 1413)

Sets the receiver of notification emails about new users.

RegistrationFeedback (page 1415)

Sets the kind of feedback that is sent to users that have registered.

RequireUniqueEmail (page 1405)

Sets if all users must have unique email addresses when registering.

UpdateHash (page 1407)

Sets if eZ Publish should update hashes if you have changed the HashType setting

UserClassGroupID (page 1410)

Sets which classes you can create in the user section in the administration interface.

UserClassID (page 1412)

Sets the class to use for user registration

UserCreatorID (page 1409)

Sets the user that will be set as the creator of self registering users.

UserGroupClassID (page 1411)

Sets the class ID of the class that represents user groups.

UseSpecialCharacters (page 1404)

Sets if special characters are allowed in passwords.

VerifyUserEmail (page 1414)

Sets if new users have to verify their account by email.

ExtensionDirectory

Summary

Sets the extensions that contain login handlers

Usage

```
ExtensionDirectory[] = extension_name1  
ExtensionDirectory[] = extension_name2  
...
```

Description

It is possible to provide custom login handlers using the extension system. This setting specifies the extensions that contain a login handler. eZ Publish will look for the handlers in the path: extension/ *extension_name*/login_handler/

LoginHandler

Summary

Sets the various methods eZ Publish will try to authenticate user logins.

Usage

```
LoginHandler[] = handler1  
LoginHandler[] = handler2  
...
```

Description

eZ Publish will try to authenticate users using the login handlers in the order they are specified. eZ Publish provides the following login handlers:

- **standard** - The default login handler for eZ Publish. Users are authenticated using the user objects found in eZ Publish itself.
- **LDAP** - The LDAP login handler. Users are authenticated through an LDAP server. Settings related to the LDAP login handler can be found in [ldap.ini](#) (page 1195).
- **textfile** - The textfile login handler allows users to be specified in a textfile similar to the `passwd` file on *NIX systems. Settings related to the textfile login handler can be found in [textfile.ini](#) (page 1430).

Common for the login handlers different from the standard login handler is that eZ Publish automatically creates these users in eZ Publish itself. These users can then log in using the standard log in handler. Users created this way are updated automatically using cronjobs.

Examples

```
LoginHandler[] = standard  
LoginHandler[] = LDAP
```

This configuration will try to log in users using the standard handler. If the standard handler fails, eZ Publish will try to authenticate the user via LDAP. If that also fails, the login is unsuccessful.

LogoutRedirect

Summary

Sets which page to redirect to when a user has logged out.

Usage

```
LogoutRedirect= URL
```

Examples

```
LogoutRedirect=/user/login
```

The default setting redirects the user to the log in page when logging out.

UseSpecialCharacters

Summary

Sets if special characters are allowed in passwords.

Usage

```
UseSpecialCharacters=true|false
```

Description

If UseSpecialCharacters is set to false, eZ Publish will only accept passwords with characters in the range a-z, A-Z and 0-9.

RequireUniqueEmail**Summary**

Sets if all users must have unique email addresses when registering.

Usage

```
RequireUniqueEmail=true|false
```

Description

We highly recommend setting RequireUniqueEmail to true.

AuthenticateMatch

Summary

Sets the fields that are accepted for login authentication.

Usage

```
AuthenticateMatch= type1 [; type2]
```

Description

Types accepted are

- **login** - The username field can be used identify a user.
- **email** - The email field can be used to identify a user. If your site allows several users with the same email address, all the users with this email address will be tested for the correct password.

In order to log in successfully the password field must always be provided.

Examples

```
AuthenticateMatch=login;email
```

The default setting allows users to log in using both the email address or the username.

```
AuthenticateMatch=login
```

This setting allows users to log in using only the username.

UpdateHash

Summary

Sets if eZ Publish should update hashes if you have changed the HashType setting

Usage

```
UpdateHash=true|false
```

Description

If this setting is enabled and eZ Publish discovers a password stored with one of the other hash types the hash is automatically converted. Note that you can't change between any of the md5 settings and the plaintext setting.

HashType

Summary

Sets the algorithm used to encrypt the user passwords stored in the database.

Usage

```
HashType=md5_password|md5_user|md5_site|plaintext
```

Description

You can choose between several different md5 algorithms and plaintext storage. We strongly recommend using one of the md5 algorithms since plaintext storage imposes a huge security risk for your users.

- **md5_password** - A hash is generated based solely on the users password.
- **md5_user** - A hash is generated based on the username and the password.
- **md5_site** - A hash is generated based on the sitename, username and password. Note that if you change the sitename your users can not log on anymore.
- **plaintext** -

UserCreatorID**Summary**

Sets the user that will be set as the creator of self registering users.

Usage

```
UserCreatorID= userid
```

Description

Users that register themselves will have a user object generated by eZ Publish. This setting controls which user on the system that will be considered the creator of the user objects.

Note that *userid* is equal to the ID number of the actual object that represents the user account.

UserClassGroupID

Summary

Sets which classes you can create in the user section in the administration interface.

Usage

```
UserClassGroupID= class_group_id
```

Description

This setting controls the classes that are listed in the "Create here" dropdown menu in the users section. You must specify a class group that contains all the user types and group types that you want to create.

Note that there can be only one usergroup class and that all classes representing users must have the ezuser datatype.

UserGroupClassID**Summary**

Sets the class ID of the class that represents user groups.

Usage

```
UserClassGroupID= classid
```

Description

There can only be one class representing user groups on your system.
Note that classid is the ID number of the class (not the identifier of the class).

UserClassID**Summary**

Sets the class to use for user registration

Usage

```
UserClassID= classid
```

Description

The UserClassID must specify a content class containing the ezuser datatype. This class will be used when new users are registering on your site.

Note that classid is the ID number of the class (not the identifier of the class).

RegistrationEmail

Summary

Sets the receiver of notification emails about new users.

Usage

```
RegistrationEmail= email_address
```

Description

If RegistrationEmail is left empty the setting [MailSettings], AdminEmail will be used instead.

Examples

```
RegistrationEmail=no@spam.org
```

eZ Publish will send notification emails about new users to "no@spam.org".

VerifyUserEmail

Summary

Sets if new users have to verify their account by email.

Usage

```
VerifyUserEmail=enabled|disabled
```

Description

If enabled, new users can't log in to your site before they have activated their account. An email will be sent to the new users with a link that activates their user.

RegistrationFeedback

Summary

Sets the kind of feedback that is sent to users that have registered.

Usage

```
RegistrationFeedback=email
```

Description

email is the only feedback type supported at the moment.

DefaultSectionID**Summary**

Sets the section ID of self registered users.

Usage

```
DefaultSectionID= number
```

Description

This setting controls the section ID assigned to new self registering users. If DefaultSectionID is set to "0", new self registering users will get the section id from the parent object set by the DefaultUserPlacement (page [1417](#)) setting.

DefaultUserPlacement**Summary**

Sets the parent object of all users who register themselves

Usage

```
DefaultUserPlacement= nodeid
```

Description

DefaultUserPlacement should be set to the ID of the node that should hold all self registered users.

AnonymousUserID

Summary

Sets the eZ Publish user that should be used for anonymous page requests.

Usage

```
AnonymousUserID= userid
```

Description

When a user that is not logged in is accessing an eZ Publish installation, the request will be handled with the permissions of the anonymous user. You can select any user on your system to represent the anonymous user.

Take care to give the correct permissions to the anonymous user since these permissions are available for all users on your site.

Note that *userid* is equal to the ID number of the actual object that represents the user account.

GeneratePasswordLength

Summary

Sets the length of passwords generated by eZ Publish.

Usage

```
GeneratePasswordLength= number
```

Description

This setting controls the length of passwords generated if GeneratePasswordIfEmpty (page 1420) is set to true

GeneratePasswordIfEmpty

Summary

Sets if eZ Publish should generate passwords if the password field is empty.

Usage

```
GeneratePasswordIfEmpty=true|false
```

Description

If GeneratePasswordIfEmpty is set to true, eZ Publish will automatically accept and generate passwords for users if the password field is empty. You can simply leave out the password field in the user register template.

If GeneratePasswordIfEmpty is set to false the user must select a password that is at least three characters long. Shorter passwords will not be accepted.

5.13.33 soap.ini

5.13.34 staticcache.ini

The configuration blocks are documented in the following sections:

- [CacheSettings] (page [1423](#))

[CacheSettings]**AlwaysUpdateArray** (page 1424)

Sets the static caches that should always be updated when an object is published.

CachedURLArray (page 1425)

Sets the content that should be statically cached.

HostName (page 1428)

Sets the hostname of the machine generating the static content.

MaxCacheDepth (page 1426)

Sets the number of levels that will be statically cached relative to the root of your installation.

StaticStorageDir (page 1427)

Sets the directory where the static cache is stored.

AlwaysUpdateArray

Summary

Sets the static caches that should always be updated when an object is published.

Usage

```
AlwaysUpdateArray[] = url1  
AlwaysUpdateArray[] = url2  
...
```

Description

Each line specifies the url of a node that will have its static cache removed whenever an object is updated on the system.

In addition to this the static cache of each node is updated whenever the corresponding object is updated.

CachedURLArray

Summary

Sets the content that should be statically cached.

Usage

```
CachedURLArray []= url1 [*]  
CachedURLArray []= url2 [*]  
...  
...
```

Description

Each line specifies one part of the content tree that will be statically cached. You can specify single nodes or complete subtrees using a wildcard (*). You must specify the full URL of the nodes, not the node names.

Note that the wildcard literally matches all characters. E.g the rule ”/weblog*” will match both ”/weblog/hello” and ”/weblogs/hello”.

Examples

```
CachedURLArray []=/  
CachedURLArray []=/news*  
CachedURLArray []=/weblog*
```

This setup will statically cache the root node and all nodes under the node with the URL news and all nodes under the node with the URL weblog.

MaxCacheDepth

Summary

Sets the number of levels that will be statically cached relative to the root of your installation.

Usage

```
MaxCacheDepth= number
```

StaticStorageDir**Summary**

Sets the directory where the static cache is stored.

Usage

```
StaticStorageDir= directory_name
```

Description

StaticStorageDir is set relative to the root directory of your eZ Publish installation. Note that you must adjust your rewrite rules if you change this setting.

Examples

```
StaticStorageDir=static
```

This setup will make eZ Publish store the statically cached files in "/static".

HostName**Summary**

Sets the hostname of the machine generating the static content.

Usage

```
Hostname= hostname [: portnumber]
```

Description

HostName sets the host of the server that serves the uncached pages. The static cache feature uses this to retrieve the generated content if there is no cache available.

Examples

```
HostName=localhost
```

The server that generates the pages is available on localhost.

5.13.35 template.ini

5.13.36 textfile.ini

5.13.37 texttoimage.ini

5.13.38 toolbar.ini

5.13.39 transform.ini

5.13.40 units.ini

5.13.41 upload.ini

5.13.42 viewcache.ini

5.13.43 webdav.ini

5.13.44 wordtoimage.ini

5.13.45 workflow.ini

You can find an overview over the workflow system in the workflows (page 163) concept chapter.

The configuration blocks are documented in the following sections:

- [EventSettings] (page 1445)
- [OperationSettings] (page 1443)
- [SimpleShippingWorkflow] (page 1440)

[SimpleShippingWorkflow]**ShippingCost** (page 1442)

Sets the cost of the shipping.

ShippingDescription (page 1441)

Sets the description that will appear on the order when using the simle shipping workflow.

ShippingDescription**Summary**

Sets the description that will appear on the order when using the simle shipping workflow.

Usage

```
ShippingDescription= description
```

ShippingCost**Summary**

Sets the cost of the shipping.

Usage

```
ShippingCost= number
```

Description

Adds the specified amount to orders if the workflow is enabled.. The currency is the same as the currency of the order.

Note: This workflow does not work with multilangauge sites.

[OperationSettings]**AvailableOperations** (page 1444)

Sets the triggers that are available from the administration interface.

AvailableOperations

Summary

Sets the triggers that are available from the administration interface.

Usage

```
AvailableOperations=[before|after_] module1_trigger1 [;[before|after_] module2_
trigger2 [;...]]
```

Description

The AvailableOperations specifies the triggers that are available in the system. Triggers are specified per module and are given a trigger name. Additionally, each trigger point can have both a before and after trigger. Each section of AvailableOperations specifies one module and a trigger name. Additionally you can specify if the trigger is a before trigger by adding before or after in front of the module name. If the trigger is both a before and an after trigger, simply don't specify anything.

eZ publish will look for the file operation_definition.php in the module directory. Each trigger activated in this setting must be properly set up in the operation_definition.php of the corresponding module.

Examples

```
AvailableOperations=content_publish;before_shop_confirmorder;shop_checkout
```

These settings will enable and make eZ publish search for the following triggers:

Module	Trigger name	Trigger time
content	publish	before
content	publish	after
shop	confirmorder	before
shop	checkout	before
shop	checkout	after

[EventSettings]**AvailableEventTypes** (page 1446)

Sets the name of the event types that eZ publish should search for.

ExtensionDirectories (page 1447)

Sets the extensions that contain event types.

RepositoryDirectories (page 1448)

Sets the directories where eZ publish will search for event types.

AvailableEventTypes

Summary

Sets the name of the event types that eZ publish should search for.

Usage

```
AvailableEventTypes []= event_group1_ event_name1  
AvailableEventTypes []= event_group2_ event_name2  
...
```

Description

This setting contains a list of event types. These event types consist of an event group and an event name. These two items are separated with an underscore (_).

The combination of the settings ExtensionDirectories (page 1447), and AvailableEventTypes specify where eZ publish will look for additional event types.

eZ publish will search for the extensions in ”/extension/ *extension_name*/eventtypes/ *event_group*/*event_name*type.php” where *extension_name* is specified by ExtensionDirectories (page 1447) and *event_group* and *event_name* are specified by AvailableEventTypes.

Examples

```
ExtensionDirectories []=ezpaynetdirect  
AvailableEventTypes []=event_ezapprove  
AvailableEventTypes []=paynet_ezpaynetdirect
```

These settings will make eZ publish search for the following files for the events.
extensions/ezpaynetdirect/eventtypes/event/ezapprove/ezapprove.php
extensions/ezpaynetdirect/eventtypes/paynet/ezpaynetdirect/ezpaynetdirecttype.php

ExtensionDirectories

Summary

Sets the extensions that contain event types.

Usage

```
ExtensionDirectories []= extension1  
ExtensionDirectories []= extension2  
...
```

Description

eZ publish will automatically search the extensions specified by ExtensionDirectories for event types. By default eZ publish will search inside the "eventtypes" directory inside your extension. The exact location of the events in this directory is specified with the AvailableEventTypes (page 1446) setting.

Most frequently ExtensionDirectories is specified in the workflow.ini.append inside custom extensions to indicate that the extensions has custom event types. This makes eZ publish recognize the events as soon as the extension is enabled.

Examples

```
ExtensionDirectories []=ezpaynetdirect
```

The ezpaynetdirect extension uses a workflow which is located in: "/extension/ezpaynetdirect/eventtypes/ AvailableEventTypes"

This setting makes eZ publish search for event types in "/extension/ezpaynetdirect/eventtypes/ AvailableEventTypes".

The AvailableEventTypes are described in the AvailableEventTypes (page 1446) setting.

RepositoryDirectories

Summary

Sets the directories where eZ publish will search for event types.

Usage

```
RepositoryDirectories[] = directory1
RepositoryDirectories[] = directory2
...
...
```

Description

The RepositoryDirectories specifies a list of directories where built in workflow event types can be found. The exact location of the workflow in this directory is specified with the AvailableEventTypes (page 1446) setting.

If you have an extension with a workflow, then use the ExtensionDirectories (page 1447) setting to register the event types.

Note: Don't change this setting unless you know what you are doing.

5.14 Libraries

ezdb (page 1450)

Provides a database abstraction layer.

ezdbschema (page 1451)

Provides a cross database schema checker and update tool.

ezfile (page 1452)

Provides cross platform file and compression utilities.

ezi18n (page 1453)

Provides functionality for reading ".ts" files and for translating text in templates.

ezimage (page 1454)

Provides an abstracted image manipulation interface.

ezlocale (page 1455)

Provides functionality for localizing dates, currencies, etc.

ezpdf (page 1456)

Provides a solution for using the template system to generate PDFs.

ezsoap (page 1457)

Provides a low level interface for the SOAP protocol.

eztemplate (page 1458)

Provides the template interpreter, compiler and the basic functions and operators.

ezutils (page 1459)

Provides small utilities (ini file parser, MIME type handlers, E-mail handling, etc.).

ezwebdav (page 1460)

Provides a low level communication interface for the WebDAV protocol.

ezxml (page 1461)

Provides low level XML DOM parsing and construction utilities.

5.14.1 ezdb

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.2 ezdbschema

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.3 ezfile

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.4 ezi18n

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.5 ezimage

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.6 ezlocale

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.7 ezpdf

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.8 ezsoap

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.9 eztemplate

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.10 ezutils

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.11 ezwebdav

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.14.12 ezxml

The eZ Publish libraries will be documented as soon as they have been reorganized and refurbished (within the end of 2005).

5.15 XML tags

The "XML block" (page 329) datatype supports the following tags / elements:

- Headings (page 331)
- Bold text (page 332)
- Italic text (page 333)
- Unformatted text (page 334)
- Lists (page 335)
- Tables (page 336)
- Hyperlinks (page 337)
- Anchors (page 339)
- Object embedding (page 340)
- Custom tags (page 342)
- Paragraphs (page 343)

S
h
y
e
u
k
r
information