

Website@School Developer Documentation 0.90.0



Contents

Package wascore Procedural Elements	2
admin.php	2
config-example.php	3
cron.php	7
file.php	8
Define WASENTRY	8
index.php	9
init.php	10
Function diff_microtime	10
Function error_exit	10
Function initialise	11
Function was_version_check	11
admin.php	13
loginlib.php	14
was.php	15
admin.php	16
loginlib.php	17
was.php	18
accountmanagerlib.php	19
Define GROUPMANAGER_DIALOG_ADD	19
Define GROUPMANAGER_DIALOG_CAPACITY_ADMIN	19
Define GROUPMANAGER_DIALOG_CAPACITY_INTRANET	19
Define GROUPMANAGER_DIALOG_CAPACITY_PAGEMANAGER	19
Define GROUPMANAGER_DIALOG_DELETE	19
Define GROUPMANAGER_DIALOG_EDIT	19
Define TASK_ACCOUNTS	19
Define TASK_GROUPS	19
Define TASK_GROUP_ADD	19
Define TASK_GROUP_CAPACITY_ADMIN	19
Define TASK_GROUP_CAPACITY_INTRANET	19
Define TASK_GROUP_CAPACITY_MODULE	19
Define TASK_GROUP_CAPACITY_OVERVIEW	19
Define TASK_GROUP_CAPACITY_PAGEMANAGER	19
Define TASK_GROUP_CAPACITY_SAVE	19
Define TASK_GROUP_DELETE	19
Define TASK_GROUP_EDIT	19
Define TASK_GROUP_SAVE	19
Define TASK_GROUP_SAVE_NEW	19
Define TASK_USERS	19
Define TASK_USER_ADD	20
Define TASK_USER_ADMIN	20
Define TASK_USER_ADVANCED	20
Define TASK_USER_DELETE	20

Define TASK_USER_EDIT	20
Define TASK_USER_GROUPADD	20
Define TASK_USER_GROUPDELETE	20
Define TASK_USER_GROUPS	20
Define TASK_USER_GROUPSAVE	20
Define TASK_USER_INTRANET	20
Define TASK_USER_MODULE	20
Define TASK_USER_PAGEMANAGER	20
Define TASK_USER_SAVE	20
Define TASK_USER_SAVE_NEW	20
Define TASK_USER_TREEVIEW	20
Define USERMANAGER_DIALOG_ADD	20
Define USERMANAGER_DIALOG_ADMIN	20
Define USERMANAGER_DIALOG_DELETE	20
Define USERMANAGER_DIALOG_EDIT	20
Define USERMANAGER_DIALOG_INTRANET	20
Define USERMANAGER_DIALOG_PAGEMANAGER	20
Function job_accountmanager	20
Function show_accounts_intro	20
Function show_accounts_menu	20
aclmanager.class.php	22
Define ACL_LEVEL_AREA	22
Define ACL_LEVEL_NONE	22
Define ACL_LEVEL_PAGE	22
Define ACL_LEVEL_SECTION	22
Define ACL_LEVEL_SITE	22
Define ACL_TYPE_ADMIN	22
Define ACL_TYPE_INTRANET	22
Define ACL_TYPE_MODULE	23
Define ACL_TYPE_PAGEMANAGER	23
area.class.php	24
Define AREA_CONST_TABLE_AREAS	24
Define AREA_CONST_TABLE_NODES	24
areamanager.class.php	25
Define AREAMANAGER_CHORE_ADD	25
Define AREAMANAGER_CHORE_DELETE	25
Define AREAMANAGER_CHORE_EDIT	25
Define AREAMANAGER_CHORE_EDIT_THEME	25
Define AREAMANAGER_CHORE_RESET_THEME	25
Define AREAMANAGER_CHORE_SAVE	25
Define AREAMANAGER_CHORE_SAVE_NEW	25
Define AREAMANAGER_CHORE_SET_DEFAULT	25
Define AREAMANAGER_CHORE_VIEW	25
Define AREAMANAGER_DIALOG_ADD	25
Define AREAMANAGER_DIALOG_DELETE	25
Define AREAMANAGER_DIALOG_EDIT	25
Define AREAMANAGER_DIALOG_EDIT_THEME	25
Define AREAMANAGER_DIALOG_RESET_THEME	25
configassistant.class.php	26

configurationmanagerlib.php	27
Define CHORE_SAVE	27
Define TASK_ALERTS	27
Define TASK_AREAS	27
Define TASK_CONFIGURATION_INTRO	27
Define TASK_SITE	27
Function job_configurationmanager	27
Function process_task_site	27
Function show_configuration_intro	28
Function show_configuration_menu	28
databaselib.php	29
Function database_factory	29
Function db_bool_is	30
Function db_delete	30
Function db_delete_sql	31
Function db_errormessage	31
Function db_escape_and_quote	31
Function db_insert_into	32
Function db_insert_into_and_get_id	32
Function db_insert_into_sql	33
Function db_last_insert_id	33
Function db_select_all_records	34
Function db_select_single_record	34
Function db_select_sql	35
Function db_update	36
Function db_update_sql	36
Function db_where_clause	37
mysql.class.php	38
Define SQL_FALSE	38
Define SQL_TRUE	38
dbsessionlib.php	39
Function dbsession_close	40
Function dbsession_create	40
Function dbsession_destroy	41
Function dbsession_exists	41
Function dbsession_expire	41
Function dbsession_garbage_collection	42
Function dbsession_get_session_id	42
Function dbsession_open	43
Function dbsession_read	43
Function dbsession_remove_obsolete_sessions	43
Function dbsession_setup	44
Function dbsession_write	44
dialoglib.php	46
Define ATTR_CLASS_ERROR	49
Define ATTR_CLASS_VIEWONLY	49
Define BUTTON_CANCEL	49
Define BUTTON_DELETE	49
Define BUTTON_GO	49

Define BUTTON_NO	49
Define BUTTON_OK	49
Define BUTTON_SAVE	49
Define BUTTON_YES	49
Define F_ALPHANUMERIC	49
Define F_CHECKBOX	49
Define F_DATE	49
Define F_DATETIME	49
Define F_FILE	49
Define F_INTEGER	49
Define F_LISTBOX	49
Define F_PASSWORD	49
Define F_RADIO	49
Define F_REAL	49
Define F_RICHTEXT	49
Define F_SUBMIT	49
Define F_TIME	49
Function accesskey_from_string	49
Function accesskey_tilde_to_underline	50
Function dialog_buttondef	50
Function dialog_get_class	51
Function dialog_get_label	51
Function dialog_get_widget	51
Function dialog_get_widget_file	52
Function dialog_get_widget_listbox	53
Function dialog_get_widget_radiocheckbox	54
Function dialog_get_widget_richtextinput	55
Function dialog_get_widget_submit	56
Function dialog_get_widget_textinput	57
Function dialog_quickform	58
Function dialog_validate	58
Function valid_datetime	59
email.class.php	61
filelib.php	62
Function get_mediatype	62
Function get_mimetype	62
Function get_mimetypes_array	63
filemanager.class.php	65
Define PARAM_FILENAME	65
Define PARAM_FILENAMES	65
Define PARAM_PATH	65
Define PARAM_SORT	65
Define SORTBY_DATE_ASC	65
Define SORTBY_DATE_DESC	65
Define SORTBY_FILE_ASC	65
Define SORTBY_FILE_DESC	65
Define SORTBY_NONE	65
Define SORTBY_SIZE_ASC	65
Define SORTBY_SIZE_DESC	65

Define TASK_ADD_DIRECTORY	65
Define TASK_ADD_FILE	65
Define TASK_CHANGE_DIRECTORY	65
Define TASK_LIST_DIRECTORY	65
Define TASK_PREVIEW_FILE	65
Define TASK_REMOVE_DIRECTORY	65
Define TASK_REMOVE_FILE	65
Define TASK_REMOVE_MULTIPLE_FILES	65
Define THUMBNAIL_PREFIX	65
groupmanager.class.php	67
Define GROUPMANAGER_MAX_CAPACITIES	67
htmllib.php	68
Function href	68
Function html_a	68
Function html_attributes	69
Function html_form	69
Function html_form_close	70
Function html_img	70
Function html_input_radio	70
Function html_input_select	71
Function html_input_submit	71
Function html_input_text	71
Function html_table	71
Function html_table_cell	72
Function html_table_cell_close	72
Function html_table_close	72
Function html_table_head	72
Function html_table_head_close	72
Function html_table_row	72
Function html_table_row_close	72
Function html_tag	72
language.class.php	74
loginlib.php	75
Define BY_EMAIL	76
Define BY_LAISSEZ_PASSER	77
Define BY_PASSWORD	77
Define LOGIN_DEBUG	77
Define LOGIN_FAILURE_DELAY_SECONDS	77
Define LOGIN_PROCEDURE_BLACKLIST	77
Define LOGIN_PROCEDURE_CHANGE_PASSWORD	77
Define LOGIN_PROCEDURE_MESSAGE_BOX	77
Define LOGIN_PROCEDURE_NORMAL	77
Define LOGIN_PROCEDURE_SEND_BYPASS	77
Define LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER	77
Define LOGIN_PROCEDURE_SHOWLOGIN	77
Define MAXIMUM_LINE_LENGTH	77
Define MINIMUM_PASSWORD_DIGITS	77
Define MINIMUM_PASSWORD_LENGTH	78
Define MINIMUM_PASSWORD_LOWERCASE	78

Define MINIMUM_PASSWORD_UPPERCASE	78
Function acceptable_new_password	78
Function authenticate_user	79
Function login_change_password	80
Function login_dialog_close	80
Function login_dialog_home_forgot_password	80
Function login_dialog_instruction	81
Function login_dialog_open	81
Function login_dialog_password_input	82
Function login_dialog_submit_input	82
Function login_dialog_text_input	82
Function login_failure_blacklist_address	83
Function login_failure_delay	83
Function login_failure_increment	83
Function login_failure_reset	84
Function login_is_blacklisted	84
Function login_page_close	85
Function login_page_open	85
Function login_send_bypass	85
Function login_send_confirmation	86
Function login_send_laissez_passer	87
Function login_stylesheet	87
Function password_hash	87
Function password_hash_check	88
Function password_salt	89
Function show_login	89
Function was_login	90
Function was_logout	91
module.class.php	93
modulelib.php	94
Function module_factory	94
modulemanagerlib.php	96
Function job_modulemanager	96
pagemanager.class.php	97
Define DIALOG_NODE_ADD	97
Define DIALOG_NODE_DELETE_CONFIRM	97
Define DIALOG_NODE_EDIT	97
Define DIALOG_NODE_EDIT_ADVANCED	97
Define DIALOG_NODE_EDIT_CONTENT	97
Define MODULE_NAME_DEFAULT	97
Define NODE_VISIBILIY_DEFAULT	97
Define NODE_VISIBILIY_EMBARGO	97
Define NODE_VISIBILIY_HIDDEN	97
Define NODE_VISIBILIY_VISIBLE	97
Define PARAM_TREEVIEW	97
Define TASK_ADD_PAGE	97
Define TASK_ADD_SECTION	97
Define TASK_NODE_DELETE	97
Define TASK_NODE_EDIT	97

Define TASK_NODE_EDIT_ADVANCED	97
Define TASK_NODE_EDIT_CONTENT	98
Define TASK_PAGE_PREVIEW	98
Define TASK_SAVE_CONTENT	98
Define TASK_SAVE_NEWPAGE	98
Define TASK_SAVE_NEWSECTION	98
Define TASK_SAVE_NODE	98
Define TASK_SET_DEFAULT	98
Define TASK_SUBTREE_COLLAPSE	98
Define TASK_SUBTREE_EXPAND	98
Define TASK_TREEVIEW	98
Define TASK_TREEVIEW_SET	98
Define TREE_VIEW_CUSTOM	98
Define TREE_VIEW_MAXIMAL	98
Define TREE_VIEW_MINIMAL	98
statisticslib.php	99
Function job_statistics	99
theme.class.php	100
themelib.php	101
Function theme_factory	101
toolslib.php	103
Define TASK_BACKUPTOOL	103
Define TASK_LOGVIEW	103
Define TASK_TOOLS_INTRO	103
Define TASK_TRANSLATETOOL	103
Function job_tools	103
Function show_tools_intro	103
Function show_tools_menu	104
Function task_backuptool	104
Function task_logview	104
translatetool.class.php	106
Define TRANSLATETOOL_CHORE_EDIT	106
Define TRANSLATETOOL_CHORE_LANGUAGE_ADD	106
Define TRANSLATETOOL_CHORE_LANGUAGE_EDIT	106
Define TRANSLATETOOL_CHORE_LANGUAGE_SAVE	106
Define TRANSLATETOOL_CHORE_LANGUAGE_SAVE_NEW	106
Define TRANSLATETOOL_CHORE_OVERVIEW	106
Define TRANSLATETOOL_CHORE_SAVE	106
Define TRANSLATETOOL_PARAM_DOMAIN	106
Define TRANSLATETOOL_PARAM_LANGUAGE_KEY	106
updatelib.php	107
Define TASK_UPDATE_CORE	107
Define TASK_UPDATE_MODULE	107
Define TASK_UPDATE_OVERVIEW	107
Define TASK_UPDATE_THEME	107
Function job_update	108
Function show_update_overview	108
Function update_core	108
Function update_core_2010120800	108

Function update core 2010122100	109
Function update core 2011020100	109
Function update core version	109
Function update module	109
Function update status update	110
Function update theme	110
useraccount.class.php	111
Define ACL ROLE GURU	111
Define ACL ROLE INTRANET ACCESS	111
Define ACL ROLE NONE	111
Define ACL ROLE PAGEMANAGER AREAMASTER	111
Define ACL ROLE PAGEMANAGER CONTENTMASTER	111
Define ACL ROLE PAGEMANAGER PAGEMASTER	111
Define ACL ROLE PAGEMANAGER SECTIONMASTER	111
Define ACL ROLE PAGEMANAGER SITEMASTER	111
Define JOB PERMISSION ACCOUNTMANAGER	111
Define JOB PERMISSION BACKUPTOOL	111
Define JOB PERMISSION CONFIGURATIONMANAGER	112
Define JOB PERMISSION FILEMANAGER	112
Define JOB PERMISSION GURU	112
Define JOB PERMISSION LOGVIEW	112
Define JOB PERMISSION MASK	112
Define JOB PERMISSION MODULEMANAGER	112
Define JOB PERMISSION NEXT AVAILABLE VALUE	112
Define JOB PERMISSION PAGEMANAGER	112
Define JOB PERMISSION STARTCENTER	112
Define JOB PERMISSION STATISTICS	112
Define JOB PERMISSION TOOLS	112
Define JOB PERMISSION TRANSLATETOOL	113
Define JOB PERMISSION UPDATE	113
Define PERMISSION AREA ADD PAGE	113
Define PERMISSION AREA ADD SECTION	113
Define PERMISSION AREA DROP PAGE	113
Define PERMISSION AREA DROP SECTION	113
Define PERMISSION AREA EDIT AREA	113
Define PERMISSION NODE ADD CONTENT	113
Define PERMISSION NODE ADD PAGE	113
Define PERMISSION NODE ADD SECTION	113
Define PERMISSION NODE DROP CONTENT	113
Define PERMISSION NODE DROP PAGE	113
Define PERMISSION NODE DROP SECTION	113
Define PERMISSION NODE EDIT CONTENT	113
Define PERMISSION NODE EDIT PAGE	113
Define PERMISSION NODE EDIT SECTION	113
Define PERMISSION SITE ADD AREA	113
Define PERMISSION SITE DROP AREA	113
Define PERMISSION SITE EDIT SITE	113
usermanager.class.php	114
Define GROUP SELECT ALL USERS	114

Define GROUP SELECT NO GROUP	114
waslib.php	115
Define CAPACITY CHAIR	115
Define CAPACITY CUSTOM1	115
Define CAPACITY CUSTOM2	115
Define CAPACITY CUSTOM3	115
Define CAPACITY CUSTOM4	115
Define CAPACITY CUSTOM5	115
Define CAPACITY CUSTOM6	115
Define CAPACITY CUSTOM7	115
Define CAPACITY CUSTOM8	115
Define CAPACITY CUSTOM9	115
Define CAPACITY EDITOR	115
Define CAPACITY MEMBER	115
Define CAPACITY NEXT AVAILABLE	115
Define CAPACITY NONE	115
Define CAPACITY PRINCIPAL	115
Define CAPACITY PROJECTLEAD	115
Define CAPACITY PUBLISHER	115
Define CAPACITY PUPIL	115
Define CAPACITY SECRETARY	115
Define CAPACITY TEACHER	115
Define CAPACITY TREASURER	115
Define QUASI RANDOM DIGITS	115
Define QUASI RANDOM DIGITS UPPER	115
Define QUASI RANDOM DIGITS UPPER LOWER	115
Define QUASI RANDOM HEXDIGITS	115
Function appropriate legal notices	115
Function build tree	116
Function calculate uri shortcuts	117
Function calc user related acs	118
Function capacity name	118
Function convert to type	118
Function cron send queued alerts	119
Function get area records	120
Function get parameter int	120
Function get parameter string	120
Function get properties	121
Function get requested area	121
Function get requested filename	121
Function get requested node	122
Function get unique number	122
Function get user groups	122
Function ini get int	123
Function is expired	123
Function is under embargo	124
Function javascript alert	124
Function lock record	125
Function lock record node	126

Function lock_release	127
Function lock_release_node	127
Function logger	127
Function magic_unquote	128
Function performance_get_queries	129
Function performance_get_seconds	129
Function quasi_random_string	129
Function quoted_printable	130
Function redirect_and_exit	131
Function replace_crlf	132
Function sanitise_filename	132
Function string2time	133
Function t	133
zip.class.php	135
Define ZIP_TYPE_BUFFER	135
Define ZIP_TYPE_FILE	135
Define ZIP_TYPE_NONE	135
Define ZIP_TYPE_STREAM	135
main_admin.php	136
Define JOB_ACCOUNTMANAGER	136
Define JOB_CONFIGURATIONMANAGER	136
Define JOB_FILEBROWSER	136
Define JOB_FILEMANAGER	136
Define JOB_FLASHBROWSER	136
Define JOB_IMAGEBROWSER	136
Define JOB_MODULEMANAGER	136
Define JOB_PAGEMANAGER	137
Define JOB_STARTCENTER	137
Define JOB_STATISTICS	137
Define JOB_TOOLS	137
Define JOB_UPDATE	137
Function add_javascript_popup_function	137
Function add_javascript_select_url_function	137
Function admin_continue_session	137
Function admin_login	138
Function admin_logout_and_exit	138
Function admin_show_login_and_exit	139
Function get_versioncheck_url	139
Function job_start	139
Function main_admin	140
main_cron.php	141
Function main_cron	141
main_file.php	142
Function download_source	142
Function download_source_tree	143
Function error_exit404	143
Function main_file	143
Function readfile_chunked	145
Function rfc1123date	145

Function send file from datadir	145
main_index.php	148
Function calculate area	148
Function calculate default page	149
Function calculate node id	149
Function main index	150
Function module load view	150
Function module view	151
Function update statistics	151
Function update view count	152
manual.php	153
version.php	154
Define WAS ORIGINAL	155
Define WAS RELEASE	155
Define WAS RELEASE DATE	155
Define WAS VERSION	155

[Package wascore Classes](#) 156

Class AclManager	156
Var \$acl_id	160
Var \$acl_type	161
Var \$area_view_areas_open	161
Var \$area_view_a_params	161
Var \$area_view_enabled	161
Var \$a_params_save	161
Var \$dialog	162
Var \$dialogdef	162
Var \$dialogdef_areas	162
Var \$dialogdef_areas_total	162
Var \$header	162
Var \$intro	163
Var \$output	163
Var \$pagination_a_params	163
Var \$pagination_enabled	163
Var \$pagination_limit	163
Var \$pagination_offset	164
Var \$pagination_total	164
Var \$related_acls	164
Constructor AclManager	164
Method build tree	165
Method calc_areas_total	165
Method dialog_tableform	166
Method enable_area_view	167
Method enable_pagination	167
Method get_dialogdef_admin	168
Method get_dialogdef_intranet	168
Method get_dialogdef_pagemanager	169
Method get_icon_area	169
Method get_icon_blank	170
Method get_permissions	170

Method get_permissions_areas	170
Method get_permissions_nodes_in_area	171
Method get_roles_intranet	171
Method get_roles_pagemanager	171
Method save_data	172
Method save_data_admin	172
Method save_data_intranet	172
Method save_data_pagemanager	172
Method save_data_permissions	172
Method set_action	173
Method set_dialog	173
Method set_header	173
Method set_intro	173
Method set_related_acls	174
Method show_dialog	174
Method show_dialog_admin	174
Method show_dialog_intranet	174
Method show_dialog_pagemanager	175
Method show_tree_walk	175
Class AdminOutput	176
Var \$breadcrumbs	177
Var \$content	177
Var \$dtd	177
Var \$funnel_mode	177
Var \$helptopic	178
Var \$high_visibility	178
Var \$html_head	178
Var \$http_headers	178
Var \$menu	178
Var \$messages_bottom	179
Var \$messages_inline	179
Var \$messages_top	179
Var \$pagination	179
Var \$subtitle	179
Var \$suppress_output	180
Var \$title	180
Constructor AdminOutput	180
Method add_breadcrumb	181
Method add_content	181
Method add_html_header	181
Method add_http_header	181
Method add_menu	182
Method add_message	182
Method add_meta	182
Method add_meta_http_equiv	182
Method add_pagination	182
Method add_pagination_item	183
Method add_popup_bottom	184
Method add_popup_top	184

Method add_stylesheet	184
Method get_address	184
Method get_bottomline	184
Method get_breadcrumbs	185
Method get_content	185
Method get_div_messages	185
Method get_html	186
Method get_html_head	186
Method get_lines	186
Method get_logo	187
Method get_menu	187
Method get_navigation	187
Method get_pagination	188
Method get_popups	188
Method get_quickbottom	189
Method get_quicktop	189
Method send_headers	190
Method send_output	190
Method set_funnel_mode	190
Method set_helptopic	190
Method set_suppress_output	190
Class Area	191
Var \$area_exists	191
Var \$area_id	191
Var \$area_record	191
Var \$module	192
Var \$nodes	192
Var \$node_id	192
Var \$requested_area	192
Var \$requested_node	192
Var \$table_areas_prefix	193
Var \$table_nodes_prefix	193
Var \$tree	193
Constructor Area	193
Method area_is_private	194
Method build_tree_of_nodes	194
Method calculate_default_descendant_node_id	195
Method calculate_node_id	195
Method calculate_validate_default_node_id	195
Method calculate_validate_node_id	196
Method exists	196
Method get_area_title	196
Method get_breadcrumb_anchors	197
Method get_childeren	197
Method get_node_link_href	197
Method get_node_record	198
Method get_node_title	198
Method get_theme_id	198
Method node2anchor	198

Method retrieve nodes from database	199
Method subtree is hidden	199
Class AreaManager	200
Var \$areas	200
Var \$output	200
Var \$show_parent_menu	201
Constructor AreaManager	201
Method area add	201
Method area delete	202
Method area edit	202
Method area edittheme	203
Method area overview	203
Method area resettheme	204
Method area save	204
Method area savenew	205
Method area setdefault	205
Method a_param	206
Method count existing theme properties	206
Method get dialogdef add area	206
Method get dialogdef edit area	206
Method get dialog data	207
Method get icon delete	207
Method get icon edit	207
Method get icon home	208
Method get options themes	208
Method get theme records	209
Method reset theme defaults	209
Method show dialog confirm delete	209
Method show edit menu	210
Method show parent menu	210
Method sort order new area	210
Class ConfigAssistant	210
Var \$dialogdef	215
Var \$dialogdef_hidden	215
Var \$fields	215
Var \$keyfield	216
Var \$language_domain	216
Var \$prefix	216
Var \$records	216
Var \$table	216
Var \$where	217
Constructor ConfigAssistant	217
Method get dialogdef	217
Method get extra	217
Method get options from extra	218
Method save data	218
Method show dialog	218
Class DatabaseMysql	219
Var \$db_link	219

Var \$db_name	219
Var \$db_password	219
Var \$db_server	220
Var \$db_type	220
Var \$db_username	220
Var \$debug	220
Var \$errno	220
Var \$error	221
Var \$prefix	221
Var \$query_counter	221
Constructor DatabaseMysql	221
Method close	222
Method column_definition	222
Method concat	223
Method connect	224
Method create_table	225
Method create_table_sql	225
Method drop_table	226
Method dump	226
Method escape	226
Method exec	227
Method last_insert_id	227
Method query	228
Method table_exists	228
Class DatabaseMysqlResult	228
Var \$errno	229
Var \$error	229
Var \$num_rows	229
Var \$result	229
Constructor DatabaseMysqlResult	229
Method close	230
Method fetch_all	230
Method fetch_all_assoc	230
Method fetch_row	230
Method fetch_row_assoc	230
Class Email	230
Var \$attachments	231
Var \$charset	231
Var \$eol	231
Var \$headers	232
Var \$mailcc	232
Var \$mailfrom	232
Var \$mailreplyto	232
Var \$mailto	232
Var \$max_length	233
Var \$message	233
Var \$minimal	233
Var \$subject	233
Constructor Email	233

Method add_attachment	233
Method add_mailcc	234
Method is_7bit	234
Method reset_all	235
Method rfc2047_qchar	235
Method rfc2047_qstring	236
Method rfc5322_address	238
Method rfc5322_message_id	239
Method send	239
Method set_header	241
Method set_mailfrom	241
Method set_mailreplyto	242
Method set_mailto	242
Method set_message	242
Method set_subject	243
Class FileManager	243
Var \$areas	244
Var \$current_directory	244
Var \$ext_allow_browse	244
Var \$ext_allow_upload	244
Var \$job	244
Var \$output	245
Var \$show_thumbnails	245
Var \$sort	245
Var \$usergroups	245
Var \$vpaths	245
Constructor FileManager	246
Method allowed_extensions	246
Method cmp_entries_bydate_asc	247
Method cmp_entries_bydate_desc	247
Method cmp_entries_byfile_asc	247
Method cmp_entries_byfile_desc	247
Method cmp_entries_bysize_asc	248
Method cmp_entries_bysize_desc	248
Method cmp_groups	248
Method delete_directory	249
Method delete_files	249
Method explode_path	250
Method file_url	250
Method get_dialogdef_add_files	250
Method get_entries	251
Method get_entries_areas	251
Method get_entries_groups	252
Method get_entries_root	252
Method get_entries_users	252
Method has_allowed_extension	253
Method human_readable_size	253
Method make_thumbnail	253
Method sanitise_filetype	254

Method show breadcrumbs	255
Method show dialog confirm delete directory	255
Method show dialog confirm delete files	255
Method show directories	256
Method show directories and files	256
Method show file as thumbnail	257
Method show list	258
Method show menu	259
Method sort entries	259
Method task add directory	259
Method task add file	259
Method task change directory	260
Method task list directory	260
Method task preview file	260
Method task remove directory	260
Method task remove file	260
Method task remove multiple files	261
Method unique filename	261
Method valid path	262
Method virusscan	262
Method vname	263
Method vpath	264
Class GroupManager	264
Var \$group_capacity_records	264
Var \$output	264
Var \$show_parent_menu	265
Constructor GroupManager	265
Method acl delete	265
Method add_group_capacity	265
Method areas_expand_collapse	266
Method a_params	266
Method calc_acl_id	266
Method capacity_admin	267
Method capacity_intranet	267
Method capacity_overview	267
Method capacity_pagemanager	267
Method capacity_save	268
Method delete_group_capacities_acls	268
Method get_dialogdef_add_group	268
Method get_dialogdef_edit_group	269
Method get_groupname	269
Method get_group_capacity_names	269
Method get_group_capacity_records	269
Method get_icon_delete	269
Method get_icon_edit	270
Method get_options_capacities	270
Method groups_overview	270
Method group_add	271
Method group_delete	271

Method group edit	272
Method group save	272
Method group savenew	272
Method has job permission	273
Method show breadcrumbs addgroup	273
Method show breadcrumbs group	273
Method show breadcrumbs groupcapacity	274
Method show menu group	274
Method show menu groupcapacity	275
Method show parent menu	275
Method valid group capacity	275
Class Language	275
Var \$default_domain	275
Var \$languages	276
Var \$phrases	276
Constructor Language	276
Method get active language names	276
Method get current language	276
Method get filenames to try	277
Method get languages to try	278
Method get phrase	278
Method get phrases from database	280
Method get phrases from file	280
Method reset cache	281
Method retrieve languages	281
Class Module	281
Var \$module_record	282
Var \$node_id	282
Constructor Module	282
Method get breadcrumb anchors	282
Method get content	282
Class Node	282
Var \$node_exists	283
Var \$node_path	283
Constructor Node	283
Method calculate_node	283
Method exists	284
Method get_area_id	284
Method get_node_path	284
Class PageManager	285
Var \$areas	285
Var \$area_id	285
Var \$output	285
Var \$tree	286
Constructor PageManager	286
Method build cached tree	286
Method calculate new sort order	286
Method calculate updated sort order	287
Method calc_home_id	289

Method delete_node	289
Method get_dialogdef_add_node	289
Method get_dialogdef_edit_advanced_node	290
Method get_dialogdef_edit_node	290
Method get_dialog_data_node	291
Method get_icon_delete	291
Method get_icon_edit	292
Method get_icon_home	292
Method get_icon_invisibility	293
Method get_icon_page_preview	293
Method get_icon_section	294
Method get_link_node_edit	294
Method get_module_records	295
Method get_node_id_and_ancestors	295
Method get_options_area	295
Method get_options_modules	296
Method get_options_parents	296
Method get_options_parents_walk	297
Method get_options_sort_order	297
Method lock_records_subtree	298
Method message_from_lockinfo	299
Method module_connect	299
Method module_disconnect	300
Method module_load_admin	301
Method module_save	301
Method module_show_edit	302
Method node_full_name	302
Method node_has_grandchildren	303
Method permission_add_any_node	303
Method permission_add_node	303
Method permission_delete_node	304
Method permission_edit_node	304
Method permission_edit_node_content	305
Method permission_set_default	305
Method queue_area_node_alert	306
Method save_node	307
Method save_node_new_area_mass_move	308
Method section_is_open	310
Method show_area_menu	310
Method show_dialog_delete_node_confirm	310
Method show_edit_menu	311
Method show_tree	311
Method show_treeview_buttons	312
Method show_tree_walk	313
Method task_node_add	314
Method task_node_delete	314
Method task_node_edit	315
Method task_node_edit_content	316
Method task_page_preview	316

Method task save content	317
Method task save newnode	317
Method task save node	319
Method task set default	319
Method task subtree collapse	319
Method task subtree expand	320
Method task treeview	320
Method task treeview set	320
Class Theme	321
Var \$area_id	321
Var \$area_record	321
Var \$breadcrumb_addendum	321
Var \$config	321
Var \$content	322
Var \$domain	322
Var \$dtd	322
Var \$friendly_url	322
Var \$high_visibility	323
Var \$html_head	323
Var \$http_headers	323
Var \$messages_bottom	323
Var \$messages_inline	323
Var \$messages_top	324
Var \$node_id	324
Var \$node_record	324
Var \$preview_mode	324
Var \$theme_id	324
Var \$theme_record	325
Var \$title	325
Var \$tree	325
Constructor Theme	325
Method add content	326
Method add html header	326
Method add http header	326
Method add message	326
Method add meta	326
Method add meta http equiv	327
Method add popup bottom	327
Method add popup top	327
Method add stylesheet	327
Method calc breadcrumb trail	327
Method calc tree visibility	328
Method construct tree	328
Method dump subtree	328
Method friendly bookmark	329
Method get address	329
Method get bottomline	329
Method get content	330
Method get div breadcrumbs	330

Method get_div_messages	330
Method get_html	331
Method get_html_head	331
Method get_jumpmenu	331
Method get_lines	332
Method get_logo	332
Method get_menu	332
Method get_navigation	333
Method get_popups	333
Method get_properties	333
Method get_quickbottom	333
Method get_quicklinks	334
Method get_quicktop	335
Method node2anchor	335
Method send_headers	335
Method send_output	336
Method set_preview_mode	336
Method show_tree_walk	336
Class TranslateTool	336
Var \$domains	337
Var \$languages	337
Var \$output	337
Var \$show_parent_menu	337
Constructor TranslateTool	337
Method a_param	338
Method code_highlight	338
Method diff_to_text	339
Method get_dialogdef_language	340
Method get_dialogdef_language_domain	340
Method get_domains	341
Method get_icon_edit	341
Method get_options_languages	341
Method get_strings_system	342
Method guess_row_count	342
Method languages_overview	343
Method language_add	343
Method language_edit	344
Method language_save	344
Method language_savenew	345
Method put_strings_userfile	345
Method render_translation_dialog	345
Method show_domain_menu	346
Method show_parent_menu	346
Method submit_diff_to_project	346
Method translation_edit	347
Method translation_save	347
Class Useraccount	347
Var \$acIs	350
Var \$acIs_areas	350

Var \$acls_modules	350
Var \$acls_modules_areas	350
Var \$acls_modules_nodes	350
Var \$acls_nodes	351
Var \$acl_id	351
Var \$area_permissions_from_nodes	351
Var \$editor	351
Var \$email	351
Var \$full_name	352
Var \$high_visibility	352
Var \$is_logged_in	352
Var \$language_key	352
Var \$path	352
Var \$properties	353
Var \$related_acls	353
Var \$username	353
Var \$user_id	353
Constructor Useraccount	353
Method fetch_acls_from_table	354
Method has_area_permissions	354
Method has_intranet_permissions	354
Method has_job_permissions	355
Method has_node_permissions	355
Method has_site_permissions	355
Method is_admin	355
Method is_admin_pagemanager	356
Method where_acl_id	356
Class UserManager	356
Var \$output	357
Var \$show_parent_menu	357
Constructor UserManager	357
Method areas_expand_collapse	357
Method a_params	358
Method calc_acl_id	358
Method delete_user_acl	358
Method get_dialogdef_add_user	358
Method get_dialogdef_add_usergroup	358
Method get_dialogdef_edit_user	359
Method get_editor_names	359
Method get_fname	359
Method get_icon_delete	359
Method get_icon_edit	360
Method get_icon_groupdelete	360
Method get_num_user_records	361
Method get_options_available_groups_capacities	361
Method get_user_names	362
Method get_user_records	362
Method has_job_permission	363
Method show_breadcrumbs_adduser	363

Method show_breadcrumbs_overview	363
Method show_breadcrumbs_user	363
Method show_menu_overview	364
Method show_menu_user	365
Method show_parent_menu	365
Method users_overview	365
Method user_add	366
Method user_admin	366
Method user_delete	366
Method user_edit	367
Method user_groupadd	367
Method user_groupdelete	367
Method user_groups	367
Method user_groupsave	368
Method user_intranet	368
Method user_pagemanager	368
Method user_save	369
Method user_savenew	369
Method user_save_basic	369
Class Zip	370
Var \$central_directory	372
Var \$Error	372
Var \$no_name_files	372
Var \$offset	372
Var \$zip_buffer	372
Var \$zip_comment	373
Var \$zip_filehandle	373
Var \$zip_path	373
Var \$zip_type	373
Constructor Zip	373
Method AddData	373
Method AddFile	374
Method CloseZip	374
Method dos_time_date	374
Method make_suitable_filename	375
Method OpenZipbuffer	375
Method OpenZipfile	376
Method OpenZipstream	376
Method zip_add_data	376
Method zip_add_directories	377
Method zip_error	377

Package wasinstall Procedural Elements	380
install.php	380
Define INSTALL_DIALOG_CANCELLED	380
Define INSTALL_DIALOG_CMS	380
Define INSTALL_DIALOG_COMPATIBILITY	380
Define INSTALL_DIALOG_CONFIRM	380
Define INSTALL_DIALOG_DATABASE	380
Define INSTALL_DIALOG_DONE	380

Define INSTALL_DIALOG_DOWNLOAD	380
Define INSTALL_DIALOG_FINISH	380
Define INSTALL_DIALOG_INSTALLTYPE	380
Define INSTALL_DIALOG_LANGUAGE	380
Define INSTALL_DIALOG_LICENSE	380
Define INSTALL_DIALOG_USER	381
Define PROJECT_SITE	381
Define WASENTRY	381
demodata.php	382
Function demodata	382
Function demodata_alerts	383
Function demodata_areas	383
Function demodata_sections_pages	383
Function demodata_users_groups	384
index.php	386
demodata.php	387
install.php	388
demodata.php	389
install.php	390
tabledata.php	391
tabledefs.php	392

[Package wasinstall Classes](#)

Class InstallWizard	394
Var \$license	395
Var \$messages	395
Var \$results	395
Constructor InstallWizard	395
Method appropriate_legal_notices	396
Method button	396
Method check_compatibility	396
Method check_for_nameclash	397
Method check_license	397
Method check_validation	397
Method clamscan_installed	397
Method construct_config_php	398
Method create_tables	398
Method end_session_and_redirect	398
Method errorcount_bump	399
Method errorcount_reset	399
Method fetch_license	399
Method gd_supported	399
Method get_default_install_values	399
Method get_dialogdef_cms	400
Method get_dialogdef_database	400
Method get_dialogdef_finish	400
Method get_dialogdef_installtype	401
Method get_dialogdef_language	401
Method get_dialogdef_user	401
Method get_list_of_install_languages	401

Method get_manifests	401
Method get_menu	402
Method get_options_db_type	403
Method get_page	403
Method guess_url	404
Method insert_tabledata	404
Method is_already_installed	404
Method magic_unquote	404
Method perform_installation	405
Method quasi_random_string	406
Method render_dialog	406
Method run	406
Method sanitise_filename	407
Method save_cms	407
Method save_database	407
Method save_installtype	408
Method save_language	408
Method save_user	408
Method show_dialog_cancelled	408
Method show_dialog_cms	409
Method show_dialog_compatibility	410
Method show_dialog_confirm	410
Method show_dialog_database	410
Method show_dialog_finish	411
Method show_dialog_installtype	411
Method show_dialog_language	412
Method show_dialog_license	412
Method show_dialog_user	412
Method t	413
Method validate	413
Method validate_password	414
Method write_config_php	414
Package waslang_en Procedural Elements	416
en_manifest.php	416
Package waslang_nl Procedural Elements	418
nl_manifest.php	418
Package wasmod_guestbook Procedural Elements	420
guestbook_admin.php	420
Function guestbook_connect	420
Function guestbook_disconnect	421
Function guestbook_save	421
Function guestbook_show_edit	422
guestbook.php	424
guestbook.php	425
Package wasmod_htmlpage Procedural Elements	427
htmlpage_admin.php	427
Function htmlpage_connect	427
Function htmlpage_disconnect	428

Function <code>htmlpage_save</code>	428
Function <code>htmlpage_show_edit</code>	429
htmlpage_cron.php	431
Function <code>htmlpage_cron</code>	431
htmlpage_install.php	432
Function <code>htmlpage_demodata</code>	432
Function <code>htmlpage_install</code>	433
Function <code>htmlpage_uninstall</code>	433
Function <code>htmlpage_upgrade</code>	433
htmlpage_manifest.php	434
htmlpage_search.php	435
Function <code>htmlpage_search</code>	435
htmlpage_view.php	436
Function <code>htmlpage_view</code>	436
htmlpage_tabledefs.php	437
htmlpage.php	438
htmlpage.php	439
Package <code>wasmod_htmlpage</code> Classes	440
Class <code>ModuleHtmlpage</code>	440
Method <code>get_content</code>	440
Package <code>wasmod_mypage</code> Procedural Elements	442
mypage.php	442
mypage.php	443
mypage_admin.php	444
Function <code>mypage_connect</code>	444
Function <code>mypage_disconnect</code>	444
Function <code>mypage_save</code>	445
Function <code>mypage_show_edit</code>	446
Package <code>wastheme_frugal</code> Procedural Elements	448
frugal_install.php	448
Function <code>frugal_demodata</code>	448
Function <code>frugal_install</code>	449
Function <code>frugal_uninstall</code>	449
Function <code>frugal_upgrade</code>	449
frugal_manifest.php	450
frugal.php	451
frugal.php	452
Package <code>wastheme_frugal</code> Classes	453
Class <code>ThemeFrugal</code>	453
Package <code>default</code> Procedural Elements	455
spellchecker.php	455
Function <code>error_handler</code>	455
Function <code>escape_quote</code>	455
Function <code>print_checker_results</code>	455
Function <code>print_suggs_elem</code>	455
Function <code>print_textindex_decl</code>	455
Function <code>print_textinputs_var</code>	456

Function print_words_elem	456
fckeditor.php	457
fckeditor_php4.php	458
Function FCKeditor_IsCompatibleBrowser	458
Package default Classes	459
Class FCKeditor	459
Var \$BasePath	459
Var \$Config	459
Var \$Height	459
Var \$InstanceName	459
Var \$ToolbarSet	460
Var \$Value	460
Var \$Width	460
Constructor FCKeditor	460
Method Create	460
Method CreateHtml	460
Method EncodeConfig	460
Method GetConfigFieldString	461
Method IsCompatible	461
Appendices	462
Appendix A - Class Trees	463
wascore	463
default	465
waslang_en	466
waslang_nl	466
wasinstall	466
wasmod_guestbook	466
wasmod_htmlpage	466
wasmod_mypage	466
wastheme_frugal	466
Appendix B - README/CHANGELOG/INSTALL	468
FAQ	469
TODO	469
LICENSE	473
README	473
HISTORY	473
INSTALL	474
CHANGES	474
CREDITS	495
Appendix D - Todo List	496

Package wascore Procedural Elements

admin.php

/admin.php - the main entypoint for website maintenance

This is one of the main entry points for Website@School. Other main entry points are /index.php, /cron.php, /file.php and also /program/install.php. Main entry points all define the constant WASENTRY. This is used in various include()ed files to detect break-in attempts.

This is a kickstarter for /program/main_admin.php.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: admin.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
require_once dirname(WASENTRY)."/config.php" [line 39]
```

```
require_once $CFG->progdire."/main_admin.php" [line 45]
```

config-example.php

/config-example.php - example of the main configuration file

This file demonstrates and documents the contents of the main configuration file 'config.php'. This file only contains the parameters that are necessary to make a connection to the database and to identify the location of the program directory both in the file system on the webserver and as seen through a web browser from the outside. It is also possible to switch debugging on via the optional \$CFG-debug variable. All other configuration parameters are to be found in the database.

As a rule, the configuration file is generated at installation time. It **MUST** be called 'config.php' and it **MUST** reside in the same directory as the main entry points [index.php](#), [admin.php](#), [cron.php](#) and [file.php](#).

Here is an overview of the 11 essential parameters and the 1 optional parameter kept in the global \$CFG object.

- \$CFG->db_type defines the database type.
Currently the only database type supported is 'mysql'. Maybe other databases will be supported in the future.

Default: 'mysql'

- \$CFG->db_server defines the name of the database server.
In the case of 'mysql' the format is 'hostname:port' where 'hostname' is a valid host (default 'localhost') and 'port' is a portnumber or the path to a local socket, e.g. '/var/lib/mysql/mysql.sock'. If the ':' and the portnumber are omitted, the default port 3306 is used.

Default: 'localhost'

- \$CFG->db_username holds the username to use when connecting to the server.
Default: ""
- \$CFG->db_password holds the password to use when connecting to the server.
Default: ""
- \$CFG->db_name holds the name of the database to use.
Default: 'was'

- \$CFG->prefix holds the tablename prefix.

The name of every table is prefixed with this value. This makes it possible to have two or more different instances of Website@School in the same database, simply by using a different prefix for every instance. Using a prefix ending with an underscore makes the resulting table names more readable and it also prevents mis-interpretation of a table name as an SQL keyword.

Default: 'was_'

- `$CFG->dir` is the absolute directory path of 'index.php' and 'config.php'.
The main entry points (index.php, admin.php, etc.) are located in `$CFG->dir` whereas all other program files are located in the directory tree starting in `$CFG->progd`.

Usually the path in `$CFG->dir` is the same as the path to the document root of the webserver.

Examples:

- /var/www/html (Red Hat),
- /home/exemplum/public_html (DirectAdmin),
- /home/httpd/htdocs (Open NA),
- C:\PROGRAM FILES\EASYPHP\WWW (Windows).

Default: '/home/httpd/htdocs'

- `$CFG->www` is the URI which corresponds with the directory `$CFG->dir`.
This URI is of the form `scheme://hostname:port/path`, where - `scheme` is either 'http' or 'https' - `hostname` is the name of the server - `port` is the number of the port the server uses to serve webpages - `path` is a path relative to the document root

Note that the colon followed by a port number are optional; the port number defaults to 80 for http and to 443 for https. Also note that the path is optional. If the path is omitted, the document root of 'hostname' is implied.

Examples:

- `http://www.example.com`
- `https://www.example.com`
- `http://www.example.com:81/web`
- `https://www.example.com:443` (the portnumber is superfluous here)
- `http://www.example.com:80` (the portnumber is superfluous here)

Default: (none)

- `$CFG->progd` is the absolute path to the program directory
The main entry points (index.php, admin.php, etc.) are located in `$CFG->dir` whereas all other program files are located in the directory tree starting in `$CFG->progd`.

Usually this directory is a subdirectory of the document root, or more precise: a subdirectory of `$CFG->dir`. The default name of this subdirectory is 'program'.

Default: '/home/httpd/htdocs/program'

- `$CFG->progwww` is the URI which corresponds with the directory `$CFG->progdir`. This URI is also of the form 'scheme://hostname:port/path', see the explanation for `$CFG->www` above.

As a rule this URI is `$CFG->www` followed by the relative path 'program'.

Important note: If you select different schemes for `$CFG->www` and `$CFG->progwww`, the browser of the website visitor may complain about mixing secure and insecure resources on the same page. It is best to use either 'http' or 'https' and not to mix both.

Default: (none)

- `$CFG->datadir` is the absolute path to a private directory outside the document root. This path points to a directory in which user documents are stored. This directory must be writeable for the user account which runs the webserver (often a user account named 'www-user' or 'www' or 'httpd' or 'apache'). This directory should NOT be accessible from the outside. Note that because of this there is no `$CFG->datawww` which corresponds to `$CFG->datadir`. The data directory should be located outside the document root. All files that need to be served from this directory tree will be served via the program code in 'file.php' in the directory `$CFG->dir`.

Note: some ISPs do not allow you to store data outside the document root. In that case you could use a 'difficult' directory name under the document root, e.g. '/home/httpd/htdocs/d3b07384d113edec49eaa6238ad5ff00', which makes it harder to guess the name and directly access the files in this directory via a browser.

Default: (none)

- `$CFG->debug` is a parameter to switch debugging ON
Via this optional boolean variable debugging can be switched on. If the parameter is not defined, it defaults to FALSE, see [init.php](#).

Default: (variable is not defined)

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: config-example.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

cron.php

/cron.php - the main entryptoint for processing cron jobs

This is one of the main entry points for Website@School. Other main entry points are /index.php, /admin.php, /file.php and also /program/install.php. Main entry points all define the constant WASENTRY. This is used in various include()ed files to detect break-in attempts.

This is a kickstarter for /program/main_cron.php.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: cron.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

require_once **dirname(WASENTRY)."/config.php"** *[line 39]*

require_once **\$CFG->progdire."/main_cron.php"** *[line 45]*

file.php

/file.php - the main entrypoint for serving files

This is one of the main entry points for Website@School. Other main entry points are [admin.php](#), [cron.php](#), [index.php](#) and also [install.php](#). Main entry points all define the constant `WASENTRY`. This is used in various include()ed files to detect break-in attempts.

This is a kickstarter for `/program/main_file.php`, but first we check for 'maintenance mode': if the file 'maintenance.html' exists in the current directory, we bail out and redirect the visitor to that file.

If we're NOT in maintenance mode, we read the essential configuration parameters from the file 'config.php' in the current directory and continue with `/program/main_index.php` to do the actual work.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: file.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

`WASENTRY = __FILE__` [line 40]

Valid entry points define `WASENTRY`; prevents direct access to include()'s.

`require_once $CFG->progdire."/main_file.php"` [line 55]

`require_once dirname(WASENTRY)."/config.php"` [line 49]

index.php

/index.php - the main entryptpoint for website visitors (frontpage)

This is one of the main entry points for Website@School. Other main entry points are [admin.php](#), [cron.php](#), [file.php](#) and also [install.php](#). Main entry points all define the constant WASENTRY. This is used in various include()ed files to detect break-in attempts.

This is a kickstarter for /program/main_index.php, but first we check for 'maintenance mode': if the file 'maintenance.html' exists in the current directory, we bail out and redirect the visitor to that file.

If we're NOT in maintenance mode, we read the essential configuration parameters from the file 'config.php' in the current directory and continue with /program/main_index.php to do the actual work.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: index.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

`require_once dirname(WASENTRY)."/config.php" [line 49]`

`require_once $CFG->progdire."/main_index.php" [line 55]`

init.php

/program/init.php - setup database connection, sessions, configuration, etc.

This file is included from one of the main entry points. The following subsystems and global variables are initialised:

- connection to the database
- session handler
- \$CFG
- etc.

This file is included at a fairly early stage in the process. It does not rely on any regular libraries which are include()'ed later on. That is: all relevant libraries (such as [waslib.php](#)) are included when necessary from within the function [initialise\(\)](#).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: init.php,v 1.1.1.1 2011-02-01 13:00:10 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

double function diff_microtime(\$time_start, \$time_stop) [line 317]

Function Parameters:

- *string* **\$time_start** starting time as a string (fractional seconds, space, seconds)
- *string* **\$time_stop** ending time as a string (fractional seconds, space, seconds)

Calculate the difference between two microtimes

void function error_exit(\$bare_condition_code, [\$page_title = 'Fatal Error']) [line 275]

Function Parameters:

- *string* **\$bare_condition_code** the bare condition code to report
- *string* **\$page_title** the title to show in the generated HTML-page

emergency exit of program in case there is something really, really wrong

This routine outputs a short message and a 'cryptic' condition code and exits the program. It is called when something goes horribly wrong during the early stages of running the program, e.g. the database cannot be opened or there is a version mismatch between the program code (the .php-files) and the database. The complete condition code is the WAS release number followed by a slash followed by the WAS version number followed by a slash and the bare condition code. The message ends with a link to about.html with 'Powered by' or 'Based on', depending on the WAS original flag. Note that we try to show graphics (including logo) but that we switch back to text-only if it is too early, ie. before [waslib.php](#) is included.

Here is an overview of meaning of the condition codes used.

- 010: cannot find config.php, is W@S installed at all?
- 015: cannot find program/main_XXXXX.php, is W@S installed at all?
- 020: configuration error, invalid database type
- 030: cannot connect to database, busy or configuration error?
- 040: error accessing the database, is W@S installed at all?
- 050: version mismatch, update to new version necessary
- 060: magic_quotes_sybase is On
- 070: there is no (default) node available in this (default) area
- 080: there is no area available
- 090: there is no valid theme available

The condition code is numeric because it is easier to report for non-English speaking users than a complicated English sentence. (The language files are not yet loaded when error_exit() is called).

- **TODO** do we really want to 'leak' a link to the main site?
- **Uses** [WAS_VERSION](#) - indicate internal version in 'cryptic' message
- **Uses** \$CFG

void function initialise() [line 42]

initialise the program, setup database, read configuration, etc.

bool/void function was_version_check([\$exit_on_error = TRUE]) [line 220]

Function Parameters:

- *bool* **\$exit_on_error** if TRUE, this routine only returns if versions match, if FALSE a mismatch is fatal

check version of PHP-files against version stored in database

this checks the main WAS_VERSION (of files) against \$CFG->version (database). if all is well, we return TRUE indicating both version numbers match. If there is a discrepancy it is logged and depending on parameter \$exit_on_error we either exit altogether OR we return FALSE to indicate the version mismatch.

Typical use is to call this routine near the start as follows (e.g. in [main_index.php](#) or [main_file.php](#)):

```
...
initialise();
was_version_check();
// Still here? Then version is OK
...
```

This forces an exit for the interfaces at the 'visitor' side. For the webmaster it is different: even if the versions do not match, we want to be able to login and do something about it via some sort of upgrade routine, e.g:

```
...
initialise();
if (!was_version_check(FALSE)) {
    do_upgrade();
} else {
    do_regular_admin();
}
...
```


admin.php

/program/languages/en/admin.php - translated messages for /program/admin.php (English)

This file is the 'mother-of-all-languages' file: it is the basis for all other translations. Because there are so many strings to translate, it is easy to lose track of which one is used where, etc. Also, it is sometimes hard to figure out what the purpose of a word or a phrase is without the context.

I try to make that a little easier by adding comments to this (main) language file. These comments will be collected in a separate array called `$comment`. The actual translations and texts end up in an array called `$string`. By using the same key in the `$comment` array, it is possible to add clarification. This clarification can be made visible in the translation tool, helping the translator grasping the purpose and context of the texts to translate.

Note that the order in which the texts appear in this file can also determine the order in which the strings are displayed in the translation tool. The comments 'follow' the strings rather than vice versa. It is not an error if a string doesn't have a comment.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: admin.php,v 1.1.1.1 2011-02-01 12:59:54 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

loginlib.php

/program/languages/en/loginlib.php - translated messages for login procedure and change password

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: loginlib.php,v 1.1.1.1 2011-02-01 12:59:54 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

was.php

/program/languages/en/was.php - generic translated messages

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: was.php,v 1.1.1.1 2011-02-01 12:59:54 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

admin.php

/program/languages/nl/admin.php - translated messages for /program/admin.php
(Dutch)

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: admin.php,v 1.1.1.1 2011-02-01 12:59:57 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

loginlib.php

/program/languages/nl/loginlib.php - translated messages for login procedure and change password

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: loginlib.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

was.php

/program/languages/nl/was.php - generic translated messages (Dutch)

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: was.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

accountmanagerlib.php

/program/lib/accountmanagerlib.php - accountmanager (users and groups)

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: accountmanagerlib.php,v 1.1.1.1 2011-02-01 13:00:13 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

GROUPMANAGER_DIALOG_ADD = 1 *[line 65]*

Distinguish between the various dialogs

GROUPMANAGER_DIALOG_CAPACITY_ADMIN = 14 *[line 70]*

GROUPMANAGER_DIALOG_CAPACITY_INTRANET = 13 *[line 69]*

GROUPMANAGER_DIALOG_CAPACITY_PAGEMANAGER = 15 *[line 71]*

GROUPMANAGER_DIALOG_DELETE = 3 *[line 67]*

GROUPMANAGER_DIALOG_EDIT = 2 *[line 66]*

TASK_ACCOUNTS = overview *[line 28]*

default selection for account manager: show introduction + links to users and groups

TASK_GROUPS = groups *[line 49]*

TASK_GROUP* relate to plain groups

TASK_GROUP_ADD = groupadd *[line 50]*

TASK_GROUP_CAPACITY_ADMIN = capacityadmin *[line 60]*

TASK_GROUP_CAPACITY_INTRANET = capacityintranet *[line 58]*

TASK_GROUP_CAPACITY_MODULE = capacitymodule *[line 59]*

TASK_GROUP_CAPACITY_OVERVIEW = capacityoverview *[line 57]*

TASK_GROUP_CAPACITY_* relate to group-capacity-combinations

TASK_GROUP_CAPACITY_PAGEMANAGER = capacitypagemanager *[line 61]*

TASK_GROUP_CAPACITY_SAVE = capacitysave *[line 62]*

TASK_GROUP_DELETE = groupdelete *[line 51]*

TASK_GROUP_EDIT = groupedit *[line 52]*

TASK_GROUP_SAVE = groupsave *[line 53]*

TASK_GROUP_SAVE_NEW = groupsavenew *[line 54]*

TASK_USERS = users *[line 31]*

TASK_USER* relate to user accounts

```

TASK_USER_ADD = useradd [line 32]
TASK_USER_ADMIN = useradmin [line 42]
TASK_USER_ADVANCED = useradvanced [line 35]
TASK_USER_DELETE = userdelete [line 33]
TASK_USER_EDIT = useredit [line 34]
TASK_USER_GROUPADD = usergroupadd [line 37]
TASK_USER_GROUPDELETE = usergroupdelete [line 38]
TASK_USER_GROUPS = usergroups [line 36]
TASK_USER_GROUPSAVE = usergroupsave [line 39]
TASK_USER_INTRANET = userintranet [line 40]
TASK_USER_MODULE = usermodule [line 41]
TASK_USER_PAGEMANAGER = userpagemanager [line 43]
TASK_USER_SAVE = usersave [line 45]
TASK_USER_SAVE_NEW = usersavenew [line 46]
TASK_USER_TREEVIEW = usertreeview [line 44]
USERMANAGER_DIALOG_ADD = 21 [line 73]
USERMANAGER_DIALOG_ADMIN = 34 [line 79]
USERMANAGER_DIALOG_DELETE = 23 [line 75]
USERMANAGER_DIALOG_EDIT = 22 [line 74]
USERMANAGER_DIALOG_INTRANET = 33 [line 78]
USERMANAGER_DIALOG_PAGEMANAGER = 35 [line 80]
void function job_accountmanager(&$output) [line 96]

```

Function Parameters:

- *object* **&\$output** collects the html output

main entry point for accountmanager (called from admin.php)

this routing dispatches the tasks. If a specified task is not recognised, the default task TASK_ACCOUNTS_OVERVIEW is executed. Note that the User Manager and the Group Manager are heavily interconnected. Therefore we use 1 common set of tasks and distinguish between both managers via sets of tasks, e.g. TASK_USER* point to the user manager where TASK_GROUP* lead to the group manager.

```
void function show_accounts_intro(&$output) [line 177]
```

Function Parameters:

- *object* **&\$output** collects the html output

display an introductory text for the account manager + menu

```
void function show_accounts_menu(&$output, [$current_task = NULL]) [line 227]
```

Function Parameters:

- *object* **&\$output** collects the html output
- *string* **\$current_task** indicate the current menu selection (if any)

display the account manager menu

aclmanager.class.php

/program/lib/aclmanager.class.php - dealing with access control lists

This file defines a class for dealing (edit+save but not create or delete) with lists of access control parameters. The main purpose is to allow easy editing of the many many permission bitmaps that are possible for both users and groups.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: aclmanager.class.php,v 1.1.1.1 2011-02-01 13:00:46 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

ACL_LEVEL_AREA = 3 *[line 55]*

limit available role options to pages, sections and areas (used in pagemanager permissions)

ACL_LEVEL_NONE = 0 *[line 46]*

limit available role options to 'none' and 'guru' (used in pagemanager permissions)

ACL_LEVEL_PAGE = 1 *[line 49]*

limit available role options to pages (used in pagemanager permissions)

ACL_LEVEL_SECTION = 2 *[line 52]*

limit available role options to pages and sections (used in pagemanager permissions)

ACL_LEVEL_SITE = 4 *[line 58]*

no limit on available role options (used in pagemanager permissions)

ACL_TYPE_ADMIN = 2 *[line 36]*

acl for administrator permissions

ACL_TYPE_INTRANET = 1 *[line 33]*

acl for intranet permissions

ACL_TYPE_MODULE = 4 *[line 42]*

acl for individual module permissions

ACL_TYPE_PAGEMANAGER = 3 *[line 39]*

acl for pagemanager permissions

area.class.php

/program/lib/area.class.php - taking care of areas

This file defines a class for dealing with areas.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: area.class.php,v 1.1.1.1 2011-02-01 13:00:34 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** we probably need to get rid of this file because it is not used (2010-12-07/PF)
- **License** [GNU AGPLv3+Additional Terms](#)

AREA_CONST_TABLE_AREAS = areas *[line 31]*

the bare name of the areas table (without prefix), should be a class constant but PHP4 doesn't do that

AREA_CONST_TABLE_NODES = nodes *[line 34]*

the bare name of the nodes table (without prefix), should be a class constant but PHP4 doesn't do that

areamanager.class.php

/program/lib/areamanager.class.php - taking care of area management

This file defines a class for managing areas (add, edit, delete, view).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: areamanager.class.php,v 1.1.1.1 2011-02-01 13:00:16 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
AREAMANAGER_CHORE_ADD = add [line 30]
AREAMANAGER_CHORE_DELETE = delete [line 31]
AREAMANAGER_CHORE_EDIT = edit [line 32]
AREAMANAGER_CHORE_EDIT_THEME = edittheme [line 33]
AREAMANAGER_CHORE_RESET_THEME = resettheme [line 34]
AREAMANAGER_CHORE_SAVE = save [line 36]
AREAMANAGER_CHORE_SAVE_NEW = savenew [line 35]
AREAMANAGER_CHORE_SET_DEFAULT = setdefault [line 37]
AREAMANAGER_CHORE_VIEW = view [line 29]
AREAMANAGER_DIALOG_ADD = 1 [line 39]
AREAMANAGER_DIALOG_DELETE = 2 [line 40]
AREAMANAGER_DIALOG_EDIT = 3 [line 41]
AREAMANAGER_DIALOG_EDIT_THEME = 4 [line 42]
AREAMANAGER_DIALOG_RESET_THEME = 5 [line 43]
```

configassistant.class.php

/program/lib/configassistant.class.php - dealing with lists of configuration parameters

This file defines a class for dealing (edit+save but not create or delete) with lists of configuration parameters. The main purpose is to allow easy editing of configuration of parts of the system, including the main program configuration.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: configassistant.class.php,v 1.1.1.1 2011-02-01 13:00:24 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

configurationmanagerlib.php

/program/lib/configurationmanagerlib.php - configurationmanager

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: configurationmanagerlib.php,v 1.1.1.1 2011-02-01 13:00:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
CHORE_SAVE = save [line 34]
TASK_ALERTS = alerts [line 32]
TASK_AREAS = areas [line 30]
TASK_CONFIGURATION_INTRO = intro [line 29]
TASK_SITE = site [line 31]
void function job_configurationmanager(&$output) [line 46]
```

Function Parameters:

- *object* **&\$output** collects the html output

main entry point for configurationmanager (called from /program/main_admin.php)

this routine dispatches the tasks, If the specified task is not recognised, the default task TASK_CONFIGURATION_INTRO is executed.

```
void function process_task_site(&$output) [line 168]
```

Function Parameters:

- *object* **&\$output** collects the html output

handle the editing/saving of the main configuration information

this routine handles editing of the main configuration parameters. It either displays the edit dialog or saves the modified data and shows the configuration manager introduction screen.

Note that we do NOT try to redirect the user via a header() after a succesful save. It would be

handy because this particular save action may have had impact on the global configuration, which is already read at this point. By redirecting we would make a fresh start, with the new parameters. However, we lose the easy ability to tell the user that the data was saved (via `$output->add_message()`). So, either no feedback or obsolete global config in core. Hmmmm. I settle for the feedback and the 'wrong' settings.

- **Uses** ConfigAssistant()

void function show_configuration_intro(&\$output) [line 103]

Function Parameters:

- *object* **&\$output** collects the html output

display an introductory text for the configuration manager + menu

void function show_configuration_menu(&\$output, [\$current_task = NULL]) [line 115]

Function Parameters:

- *object* **&\$output** collects the html output
- *string* **\$current_task** indicate the current menu selection (if any)

display the configuration manager menu

databaselib.php

/program/lib/database/databaselib.php - database factory and database access routines

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: databaselib.php,v 1.1.1.1 2011-02-01 13:00:52 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool/object function database_factory(\$prefix, [\$db_type = 'mysql'], [\$debug = FALSE]) [*line 64*]

Function Parameters:

- *string* **\$prefix** the tablename prefix
- *string* **\$db_type** (optional) which database to use, default 'mysql'
- *bool* **\$debug** if TRUE extra information is displayed (handy for debugging the code)

manufacture a database object

This loads (includes) a specific database access class based on the parameter \$db_type. Currently 'mysql' is the only option, but support for PostgreSQL or other databases could be added in the future, see the code that is commented out

Because Website@School is not meant to be an 'enterprisy application', I decided against using an abstract class that would be extended by a specific driver class which would be instantiated via yet another factory type class; I'd like to keep this as simple as possible while retaining the necessary flexibility (and the option to add support for other databases). Toolkits like Adodb seem overkill for this application program.

This routine is called at a fairly early stage in the process. It does not rely on any regular libraries which may be include()'ed lateron. If no valid database type is specified, the function returns FALSE, otherwise a database object is returned.

Note that I did not use a singleton because I think that that pattern is simply a fancy word for a global variable. YMMV.

Note: This file can be safely included from the [install.php](#) script, allowing for database-

manipulations via this abstraction layer rather than directly going to the database. There are no dependencies on other include()'s other than the actual database class files such as mysql.class.php. Also, this file does not rely on the global variable \$CFG, which is also very convenient in the installer (where no CFG is available).

- **TODO** perhaps add postgresql in a future version

void function db_bool_is(\$value, \$variable_to_check) [line 380]

Function Parameters:

- *bool/mixed* **\$value** value to test for, could be TRUE, FALSE or anything else
- *mixed* **\$variable_to_check** the value of the variable to check

check boolean field in a database-independent way

Various databases have different ways to indicate TRUE or FALSE in boolean type of fields. MySQL uses a tinyint(1) with values NULL, 0 and 1. PostgreSQL uses a lowercase 't' or 'f' etc. We already have two database-specific definitions for TRUE and FALSE: SQL_TRUE and SQL_FALSE. This routine 'converts' the database-specific boolean values back to a form that is useable in PHP. This routine is able to test for either TRUE or FALSE. Any other value is returned as NULL.

```
Typical use: $user = db_select_single_record('users','is_active','user_id = 13');
if (db_bool_is(TRUE,$user['is_active'])) {
    ...
}
```

bool/int function db_delete(\$tablename, [\$where = ""]) [line 336]

Function Parameters:

- *string* **\$tablename** the name of the table to delete from (without prefix)
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)

delete zero or more rows in a table

- Uses [db_delete_sql\(\)](#)

string function db_delete_sql(\$tablename, [\$where = ""]) [*line 348*]

Function Parameters:

- *string* **\$tablename** the name of the table to delete from (without prefix)
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)

generate SQL to delete zero or more rows in a table

- Usedby [db_delete\(\)](#)

string function db_errormessage() [*line 458*]

retrieve the latest database error from \$DB

- Uses \$DB;

string/mixed function db_escape_and_quote(\$value) [*line 277*]

Function Parameters:

- *mixed* **\$value** string, boolean, null or other value to escape and quote

conditionally quote and escape values for use with a database table

If \$value is a string, it is escaped and single quotes are added at begin and end. If \$value is a boolean, it is converted into the correct value for the database using SQL_FALSE/SQL_TRUE If \$value is NULL, it is converted into the string 'NULL' (without

quotes) Otherwise the value is not changed.

- See <http://xkcd.com/327>
- Usedby [db_select_sql\(\)](#)
- Uses \$DB
- Usedby [db_insert_into_sql\(\)](#)

bool function db_insert_into(\$tablename, \$fields) [line 120]

Function Parameters:

- *string* **\$tablename** the name of the table to insert into (without prefix)
- *array* **\$fields** an associative array with fieldnames and fieldvalues

execute the necessary SQL-code for an INSERT INTO statement

This excutes the SQL-statement created by [db_insert_into_sql\(\)](#).

- Uses \$DB

bool function db_insert_into_and_get_id(\$tablename, \$fields, [\$key_fieldname = ""]) [line 137]

Function Parameters:

- *string* **\$tablename** the name of the table to insert into (without prefix)
- *array* **\$fields** an associative array with fieldnames and fieldvalues
- *string* **\$key_fieldname** the name of the field that holds the primary key/serial

execute the necessary SQL-code for an INSERT INTO statement and return the last_insert_id

This executes the SQL-statement created by [db_insert_into_sql\(\)](#). If all goes well, the value of the last inserted id is returned.

- **Uses** `$DB`

string function `db_insert_into_sql($tablename, $fields)` [line 95]

Function Parameters:

- *string* **\$tablename** the name of the table to insert into (without prefix)
- *array* **\$fields** an associative array with fieldnames and fieldvalues

generate the necessary SQL-code for an INSERT INTO statement

Construct an SQL-statement that inserts data into the specified table. This routine takes care of properly escaping strings and also handles the addition of the table prefix

- **Uses** [db_escape_and_quote\(\)](#)

int|bool function `db_last_insert_id([$tablename = ''], [$fieldname = ''])` [line 409]

Function Parameters:

- *string* **\$tablename** name of the table (without prefix) in which a record was inserted
- *string* **\$fieldname** name of the serial field to examine

wrapper for DB->last_insert_id()

This calls `$DB->last_insert_id()` in a way that should be compatible with a future PostgreSQL database class. Note that MySQL doesn't care about this. You can get away with leaving table and field parameters empty (as is the default), but for compatibility and documentation purposes you should use the correct values.

Typical use: `db_insert_into('users',$fields_array);` `$user_id` =

```
db_last_insert_id('users','user_id');
```

- **Uses \$DB**

bool/array function `db_select_all_records($tablename, $fields, [$where = "], [$order = "], [$keyfield = "], [$limit = "], [$offset = "])` *[line 251]*

Function Parameters:

- *string* **\$tablename** name of the table to select from (without prefix)
- *mixed* **\$fields** fieldname or array with fieldnames to select
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)
- *mixed* **\$order** fieldname or array with fieldnames to determine sort order (without ORDER BY keyword)
- *string* **\$keyfield** field to use as the key in the returned array or empty for 0-based numeric array key
- *int* **\$limit** the maximum number of records to retrieve
- *int* **\$offset** the number of records to skip initially

fetch all selected records from the database in one array

- **Uses** [db_select_sql\(\)](#)

bool/array function `db_select_single_record($tablename, $fields, [$where = "], [$order = "])` *[line 222]*

Function Parameters:

- *string* **\$tablename** name of the table to select from (without prefix)
- *mixed* **\$fields** fieldname or array with fieldnames to select

- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)
- *mixed* **\$order** fieldname or array with fieldnames to determine sort order (without ORDER BY keyword)

fetch a single record from the database

- Uses [db_select_sql\(\)](#)

string function db_select_sql(\$tablename, \$fields, [\$where = ''], [\$order = '']) [*line 178*]

Function Parameters:

- *string* **\$tablename** name of the table to select from (without prefix)
- *mixed* **\$fields** fieldname or array with fieldnames to select
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)
- *mixed* **\$order** fieldname or array with fieldnames to determine sort order (without ORDER BY keyword)

generate the necessary SQL-code for a simple SELECT statement

Construct an SQL-statement of the form: SELECT field_list FROM table WHERE where_expression ORDER BY orderby_list

The parameter \$fields can be either a simple string, indicating a single field or an array when more fields are to be selected

The optional parameter \$where is either a simple string with an appropriate expression (without the keyword WHERE) or an array with fieldname/value-pairs. In the latter case the clauses fieldname=value are AND'ed together. If the specified values are string-like, they are properly quoted. Boolean values are treated properly too.

The optional parameter \$order is either a simple string with an appropriate list or expression (without the keyword ORDER BY) or an array with fieldnames which will be used to create a comma-delimited string.

Examples:

1. `db_select_sql('areas','title')` yields `'SELECT title FROM was_areas'`
2. `db_select_sql('areas',array('title','theme_id'),array('is_visible' => TRUE),'sort_order')` yields `'SELECT title,theme_id FROM was_areas WHERE is_visible = 1 ORDER BY sort_order'` (if `SQL_TRUE` is `'1'`)

- Usedby [db_select_all_records\(\)](#)
- Usedby [db_select_single_record\(\)](#)
- Uses [db_escape_and_quote\(\)](#)

bool/int function `db_update($tablename, $fields, [$where = ''])` [*line 300*]

Function Parameters:

- *string* **\$tablename** the name of the table to update (without prefix)
- *array* **\$fields** an associative array with fieldnames and fieldvalues
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword))

update one or more fields in a table

- Uses [db_update_sql\(\)](#)

string function `db_update_sql($tablename, $fields, [$where = ''])` [*line 314*]

Function Parameters:

- *string* **\$tablename** the name of the table to update (without prefix)
- *array* **\$fields** an associative array with fieldnames and fieldvalues
- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword))

generate sql to update one or more fields in a table

- Usedby [db_update\(\)](#)
- Uses \$DB

string function db_where_clause([\$where = ""]) [*line 431*]

Function Parameters:

- *mixed* **\$where** a single clause or an array with fieldnames => values ((without the WHERE keyword)

construct a where clause from string/array, including the word WHERE

this constructs a where clause including the word 'WHERE' based on the string or array \$where.

The optional parameter \$where is either a simple string with an appropriate expression (without the keyword WHERE) or an array with fieldname/value-pairs. In the latter case the clauses fieldname=value are AND'ed together. If the specified values are string-like, they are properly quoted. Boolean values are treated properly too. NULL-values yield a standard 'IS NULL' type of expression.

mysql.class.php

/program/lib/database/mysql.class.php - access to mysql via database class

This file is included from the poor mans database factory in [databaselib.php](#). It provides all necessary functionality to use MySQL as the underlying database server.

This file defines two classes: DatabaseMysql and DatabaseMysqlResult. Typical usage would be:

```
$DB = new Database($table_prefix);           // once at program start
$DB->connect($host,$usr,$pwd,$dbname);        // once at program start

$sql = "SELECT * FROM {$DB->prefix}foo";      // select all rows from table foo,
$DBResult = $DB->query($sql);
$all_rows = $DBResult->fetch_all_assoc()      // store everything in an array,
$DBResult->close();                          // and free the memory

...                                           // do something with $all_rows

$DB->close();                                // once at program end
```

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: mysql.class.php,v 1.1.1.1 2011-02-01 13:00:52 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

SQL_FALSE = 0 [line 48]

this circumvents the sub-optimal implementation of booleans in MySQL

SQL_TRUE = 1 [line 51]

this circumvents the sub-optimal implementation of booleans in MySQL

dbsessionlib.php

/program/lib/dbsessionlib.php - functions to keep PHP-sessions in the database

This file provides the functions to handle sessions via the database rather than via files or other standard PHP-mechanisms.

Useful information about storing sessions can be found in these user comments:
<http://php.net/manual/en/function.session-set-save-handler.php>

Important issues:

- there is an issue with writing session data when using a database class because the database class is already destroyed when PHP tries to write the session data. Workaround: call `session_commit()` (alias for `session_write_close()`) near the end of the script.
- there is a security risk when the script simply accepts any session id that is presented via a cookie; a session should exist/be created before data is written and the session key should have been created in a previous call and thus be present in the database
- different versions of PHP handle callback parameters differently where object methods are involved. The most generic way to work around these incompatibilities is to use global functions for open, close, etc. rather than methods in some session class.
- session keys should be sanitised before manipulating the database in order to prevent an SQL injection.
- `session.auto_start` may make it impossible to substitute our own session handlers if it is set in `php.ini`
- how about locking a session?

There is a difference between the maximum session duration and the session time out. A session could last for 'duration' seconds but only if there is still activity at least every timeout seconds. There should be a maximum session lifetime of say 24 hours. There also should be a timeout of say 60 minutes.

Sessions are stored in a table called 'sessions'. This table is defined as follows: `session_id` serial

`session_key` varchar(255)

`session_data` longtext

`user_id` int unsigned

`user_information` varchar(255)

`ctime` datetime

`atime` datetime

primary key(`session_id`)

foreign key(`user_id`) references users(`user_id`)

unique index(`session_key`)

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: dbsessionlib.php,v 1.1.1.1 2011-02-01 13:00:11 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **Usedby** [admin_continue_session\(\)](#)
- **License** [GNU AGPLv3+Additional Terms](#)

bool function dbsession_close() [*line 214*]

'close' a session that was opened with dbsession_open() before

Since this function has no way to tell `_which_` session should be closed, it is utterly useless (but it has to exist to satisfy `session_set_save_handler()`) The function [dbsession_open\(\)](#) has the same uselessness, so they are a perfect pair.

- **TODO** should we do something with locking the session record from `dbsession_open()` until `dbsession_close()`? For now, the session record is not locked in any way, so the latest call gets to keep its changes Mmmm....

bool|string function dbsession_create(\$user_id, [\$user_information = ""]) [*line 125*]

Function Parameters:

- *int* **\$user_id** link to the users table, identifies the user that started the session
- *string* **\$user_information** (optional) auxiliary information about the user, e.g. the IP-address

create a new session in the session table, return the unique sessionkey

this creates generates a new unique session key and stores it in a new record in the sessions table. Additional information is recorded in the new record too: the `user_id` and auxiliary information. This information makes that a session can always be linked to a particular user (which is handy when dealing with locked pages, etc.). This routine attempts to create a unique session key a number of times. If it doesn't work out, the routine returns FALSE.

the optional parameter `$user_information` can be used to store additional information about this user, e.g. the IP-address. This is useful for generating messages like 'Node xxx is

currently locked by user YYYY logged in from ZZZZ'.

Note that the generation of a unique session key is salted with both the main url of this website and the special salt that was recorded once during installation time. Also, pseudo-random data is added via `rand()`. Hopefully this will be hard to guess, even though we use `md5()` to condense this (semi-)random information into only 128 bits.

- **TODO** should we also record the IP-address of the user in the session record? In a way this is a case of information leak, even though it is only between authenticated users. Mmmm...
- **Uses** \$DB
- **Uses** \$CFG

bool function dbsession_destroy(\$session_key) [line 267]

Function Parameters:

- *string* **\$session_key** the unique session_key that identifies the session

remove a session record from the sessions table (it should still exist)

remove the specified record from the sessions table. it is an error if the record does not exist.

bool function dbsession_exists(\$session_key) [line 159]

Function Parameters:

- *string* **\$session_key** the unique session_key that identifies the session

check to see if \$session_key exists in the session table

This checks the existence of a session in the sessions table. Session keys are only generated from [dbsession_create\(\)](#). This prevents us accepting spurious session keys via a manipulated cookie. If the session key does not exist, the call fails and FALSE is returned.

bool function dbsession_expire(\$max_lifetime) [line 308]

Function Parameters:

- *int* **\$max_lifetime** the lifetime value to automatically expire sessions (in seconds)

remove all sessions that were created more than \$max_life seconds ago

not only are sessions terminated when there is no more activity for \$time_out seconds (@see [dbsession_garbage_collection\(\)](#)) but also the total lifetime of a session is limited to \$life_time seconds. This routine is not part of the required session handlers but it can be called periodically (@see [cron.php](#)).

- Uses [dbsession_remove_obsolete_sessions\(\)](#)

bool function [dbsession_garbage_collection\(\\$time_out\)](#) [*line 285*]

Function Parameters:

- *int* **\$time_out** the time-out value to automatically expire sessions (in seconds)

remove all sessions that are last accessed more than \$time_out seconds ago

- Usedby [was_login\(\)](#)
- Uses [dbsession_remove_obsolete_sessions\(\)](#)

int|bool function [dbsession_get_session_id\(\\$session_key\)](#) [*line 173*]

Function Parameters:

- *string* **\$session_key** the unique session_key that identifies the session

retrieve the session_id (pkey) that corresponds with session_key

this is very similar to [dbsession_exists\(\)](#). This routine returns the actual session_id integer, whereas [dbsession_exists\(\)](#) only returns TRUE.

bool function dbsession_open(\$save_path, \$session_name) [*line 195*]

Function Parameters:

- *string* **\$save_path** (unused) pathname relevant for file based session handler
- *string* **\$session_name** (unused) the non-unique session_name that identifies the cookie in the user's browser

'open' a session

this 'opens' a session. note that this function is unable to identify the session because it is only presented with

- the \$save_path (which is relevant only with file-based session handlers)
- the \$session_name (which is the _name_ of the session, but not the _token_)

there is no way to let this function do anything useful, so it boils down to a dummy always returning TRUE. The function [dbsession_close\(\)](#) has the same uselessness, so they are a perfect pair.

string function dbsession_read(\$session_key) [*line 224*]

Function Parameters:

- *string* **\$session_key** the unique session_key that identifies the session

read the (serialised) session data from the database

bool function dbsession_remove_obsolete_sessions(\$seconds, \$time_field) [*line 332*]

Function Parameters:

- *int* **\$seconds** the period of time after which the session is obsolete
- *string* **\$time_field** the field to use for time comparisons: either 'atime' or 'ctime'

workhorse for removing obsolete sessions from the database

this logs and subsequently removes obsolete sessions from the sessions table It is a workhorse function for both [dbsession_garbage_collection\(\)](#) and [dbsession_expire\(\)](#).

Session records are removed when the \$time_field in the sessions table contains a date/time that is older than \$seconds seconds ago. Before the records are removed, we retrieve them

and log pertinent information from each one via `logger()`, for future reference.

Note that we try to continue with deleting records, even if the logging appears to have generated errors.

- Usedby [dbsession_expire\(\)](#)
- Usedby [dbsession_garbage_collection\(\)](#)

bool function `dbsession_setup($session_name)` [line 85]

Function Parameters:

- *string* **\$session_name** the name of the session (usually 'PHPSESSID' in generic PHP-applications)

setup database based handlers for session management

this is basically shorthand for `session_set_save_handler()` this routine replaces the existing session handlers with the handlers specified below in this file.

- Usedby [was_login\(\)](#)
- Usedby [was_logout\(\)](#)

bool function `dbsession_write($session_key, $session_data)` [line 244]

Function Parameters:

- *string* **\$session_key** the unique session_key that identifies the session
- *string* **\$session_data** the string with (serialised) session variables

write the (serialised) data to the database

dialoglib.php

/program/lib/dialoglib.php - useful functions for manipulating dialogs

This file provides various utility routines for creating and validating user dialogs.

A dialog is a collection of input elements grouped together in a form. An input element (or field) has at least a type (e.g. F_ALPHANUMERIC or F_DATETIME) and a name (e.g. 'title' or 'expiry') and optionally one or more of the other possible properties. The name of an input element uniquely identifies the field in the dialog; it can be used to retrieve the data entered by the user from the global \$_POST array.

A dialog can be defined via a 0-based array, where every array element is separate input element. Each of the input elements is in itself an associative array with property-value-pairs. The recognition of a property depends on the type, e.g. the number of decimals is irrelevant for a string-type input element.

Here is an overview of properties and field types to which they apply. An 'x' means required, an 'o' means optional and a '-' means don't care.

type

	name	value	rows	columns	minlength	maxlength	decimals	minvalue	maxvalue	options	label	accesskey	alt	class	viewonly	tabindex	title	hidden	id
F_ALPHANUMERIC	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_INTEGER	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_REAL	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_DATE	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_TIME	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_DATETIME	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_PASSWORD	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
F_CHECKBOX	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o

```

F_LISTBOX    x x o o - - - - - x o o o o o o o o o
F_RADIO      x x o - - - - - x o o o o o o o o o
F_FILE       x x o - o - - - - - o o o o o o o o - o
F_SUBMIT     x x x - - - - - - - o o o o o o o - o
F_RICHTEXT   x x o o o o o o - - - - - o o o o o o o o o o

```

There are two more properties: 'errors' and 'error_messages'. These properties can be set whenever the validation of the input yields an error. If all is well, 'errors' is either not set or equal to 0.

Here is a description of the various properties.

- type:
the type of the input element, one of the F_* constants defined at the top of this file.
- name:
the name to uniquely identify the input element
- value:
the current value of the element, this value needs to be displayed in the dialog
- rows:
the number of text rows for this element. This applies only to generic text fields (alphanumerics) and listboxes.
- columns:
the number of characters to show in the input element. This applies only to text inputs (not lists or checkboxes).
- minlength:
the minimum number of characters that need to be entered by the user. If 0, the field can be left empty.
- maxlength:
the maximum number of characters that can be entered by the user. Note that this number is no necessarily the same as the number of columns to display (or even the product of columns and rows).
- decimals:
the number of decimals in the fraction of real numbers
- minvalue:
the minimum value that needs to be entered. This applies to numeric fields (integer, real) and also to dates and times.
- maxvalue:

the maximum value that can be entered. This applies to numeric fields (integer, real) and also to dates and times.

- **options:**
this is a list (array) of key-value-pairs that identify the allowable values for a listbox or radio buttons. The key is used as the fields value and the value is the descriptive title of the option.
- **label:**
this is the text that is displayed `_before_` the input element. It is used to indicate the purpose of the input element.
- **accesskey:**
this is a single letter that can be used to access the element via the keyboard by using a key combination like [Alt-A].
- **alt:**
an alternative text that describes the element (accessibility)
- **class:**
this identifies the class(es) that need to be associated with this element. This allows for changing the style of the element.
- **readonly:**
indicates that this element is not to be changed by the user but that it can be displayed nevertheless
- **tabindex:**
a number that indicates the order in which fields area accessed when using the [Tab] key to move to the next element.
- **title:**
a descriptive title that is displayed when the mouse is hovering over the element
- **hidden:**
identifies a field that should be part of the dialog, but not visible to the user. Note that by specifying a hidden list, it is possible to validate the value of the hidden input against the list of acceptable values once it returns from the user. However, one should never trust the value of a field that is sent in 'hidden' form, because after all it is still user input, no matter what.
- **id**
a unique (within the document) identifier for this input element. This id is used to link a label to an input. The id should start with a letter. The corresponding label is identified by appending the string `'_label'` to the id.

- **errors:**
the number of errors encountered after validating the value of this element
- **error_messages:**
an array of messages identifying the problems encountered with this element during validation

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: dialoglib.php,v 1.1.1.1 2011-02-01 13:00:50 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

ATTR_CLASS_ERROR = error [line 200]
 ATTR_CLASS_VIEWONLY = viewonly [line 201]
 BUTTON_CANCEL = cancel [line 204]
 BUTTON_DELETE = delete [line 206]
 BUTTON_GO = go [line 209]
 BUTTON_NO = no [line 208]
 BUTTON_OK = ok [line 203]
 BUTTON_SAVE = save [line 205]
 BUTTON_YES = yes [line 207]
 F_ALPHANUMERIC = alphanumeric [line 186]
 F_CHECKBOX = checkbox [line 193]
 F_DATE = date [line 189]
 F_DATETIME = datetime [line 191]
 F_FILE = file [line 197]
 F_INTEGER = integer [line 187]
 F_LISTBOX = listbox [line 194]
 F_PASSWORD = password [line 192]
 F_RADIO = radio [line 195]
 F_REAL = real [line 188]
 F_RICHTEXT = richtext [line 198]
 F_SUBMIT = submit [line 196]
 F_TIME = time [line 190]
 string function accesskey_from_string(\$string) [line 1373]

Function Parameters:

- *string* **\$string** the string to process

return the character that follows the first tilde in a string

this returns the character that follows the first tilde in the string (if any). This is the character that could be added as an accesskey to some HTML tag, e.g. a label or an input.

string function `accesskey_tilde_to_underline($string, [$tag_open = '<u>'], [$tag_close = '</u>'])` [line 1350]

Function Parameters:

- *string* **\$string** the string to process
- *string* **\$tag_open** the tag that starts the emphasis
- *string* **\$tag_close** the tag that ends the emphasis

replace tilde+character with emphasised character to indicate accesskey

this replaces the combination of a tilde and the character following it with \$tag_open followed by the character followed by \$tag_close.

Example: `accesskey_tilde_to_underline("~Username")` yields `"<u>U</u>sername"`
`accesskey_tilde_to_underline("Pass~word",",")` yields `"Password"`
`accesskey_tilde_to_underline("~Role",",")` yields `"Role"`

array function `dialog_buttondef($button_type, [$value = "], [$title = "])` [line 1243]

Function Parameters:

- *string* **\$button_type** one of the predefined button constants, e.g. `BUTTON_OK`
- *string* **\$value** the label used for display including hotkey, eg. `'~Yes'` or `'~Cancel'`
- *string* **\$title** the text displayed via a mouseover

shortcut for generating a dialogdef for a button

this constructs an array describing a button. The button definition is bare bones but it includes name, class, value and optionally a title. If no \$value is specified, a translated value is retrieved for the button. If no \$title is specified, a title indicating the hotkey is constructed. This more or less works around the problem that hotkeys cannot be visualised in an input type="submit" button (It is possible in a button tag, but we don't use that because it requires HTML 4. Maybe later...)

string function dialog_get_class(&\$item, [\$class = NULL]) [*line 1314*]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$class** class to start with, otherwise use \$item['class']

construct a space-delimited list of classes that apply to this item

this constructs a string with applicable classes for this element. if the item has validation errors, the class ATTR_CLASS_ERROR is added, if the item is viewonly, the class ATTR_CLASS_VIEWONLY is added. This allows for the CSS to change the style depending on these situations.

string function dialog_get_label(&\$item) [*line 295*]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element

construct a label for a dialog input element

this constructs the label for an input. It is built inside a label-tag, possibly with these attributes: id, for, accesskey, class, title. The class is a special case: if there were errors the class 'ATTR_CLASS_ERROR' is added to the class attribute, in case of viewonly the class ATTR_CLASS_VIEWONLY' is added too. (Note that the latter shouldn't happen: how can a viewonly field yield any errors at all unless someone is trying to crack the program?)

Because some browsers require that the label of a listbox is linked to the select tag (done via 'id' and 'for'), we MUST have some 'id' for that particular tag. If it is not there, we generate one and add it automatically.

- **TODO** if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

array/string function dialog_get_widget(&\$item) [*line 353*]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element

construct an actual HTML input widget for dialog input element

this constructs the actual HTML-code for a dialog input element. If the input element is 'hidden', we generate a minimalistic hidden field with no labels, accesskeys or whatever: basically just a name-value-pair to communicate to a subsequent form.

If the item is genuine (it has a name and a type), we construct the input using a workhorse routine.

- **TODO** we could manipulate the title attribute of input strings like "please enter a number between {MIN} and {MAX}" based on the various value properties instead of just displaying the title. oh well, for a future version, perhaps...
- **TODO** we now only cater for buttons via input type="submit" without the option to visualise the accesskey. Using the button tag could solve that, but button is not defined before HTML 4.01. What to do?

array/string function dialog_get_widget_file(&\$item, \$name, \$value) [line 1187]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the (current) value of the input element to show ('field value')

construct an input field for file upload

this constructs an input widget for uploading files. This usually includes a button to browse the user's local file system (depends on browser).

Note that it is NOT possible to 'preload' a value in this input field; any predefined value is ignored by the browser. As a workaround we could show the \$value to the user using an additional comment, e.g. by adding it to the label or something. For now, we simply do nothing with the value.

The properties recognised translate to the following HTML-code/attributes
 name : name
 value : value (see note above)
 accesskey : accesskey

columns : cols (textarea) or size (input type="text")
 alt : alt
 class : class (also depends on viewonly and errors)
 tabindex : tabindex
 id : id
 title : title
 viewonly : disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
 errors : addition of ATTR_CLASS_ERROR to class list (if errors > 0)

- **TODO** if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?
- **TODO** should we do something with an um-empty \$value? If so, waht? The browser ignores this...

array function dialog_get_widget_listbox(&\$item, \$name, \$value, \$f_type) [line 997]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the (current) value of the input element to show ('field value')
- *string* **\$f_type** the type of the field (eg text, number, date, time, ...)

construct a listbox

this constructs a listbox

The number of lines in the result depends on the number of items in the options array in \$item. The result always starts with a SELECT opening tag, followed by N OPTION tags and finally a SELECT closing tag.

There are two different ways to specify the options. The simple way is to have a single options array with 'value' => 'option text' pairs. In this case the available properties such as title, class and viewonly are copied from the corresponding generic properties in the \$item array,

The other way is to have an array of arrays like this:

```
$item['options'] = array('1'=>array('title'=>'...', 'option'=>'...'), 2 => array(...));
```

This allows for setting properties of individual options, e.g. one of the options could be made viewonly while the others are still selectable.

The properties recognised translate to the following HTML-code/attributes name : name
value : value AND perhaps 'selected' if value matches option value
accesskey : accesskey
alt : alt
class : class (also depends on viewonly and errors)
tabindex : tabindex
id : id
title : title
viewonly : disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
errors : addition of ATTR_CLASS_ERROR to class list (if errors > 0)
label : tilde+letter may change the accesskey

Note that the options within the SELECT tag are indented 2 spaces for readability.

array/string function dialog_get_widget_radiocheckbox(&\$item, \$name, \$value, \$f_type) [line 873]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the (current) value of the input element to show ('field value')
- *string* **\$f_type** the type of the field (eg text, number, date, time, ...)

construct a checkbox or 1 or more radiobuttons

this constructs a checkbox or a list of radiobuttons.

Note: because a checkbox and radionbuttons are very similar, they are handled in the same workhorse routine. Maybe we should split this in the name of code clarity. Oh well...

If we are generating a checkbox, the result looks something like this: <input type="checkbox" value="1" checked ...><label ...>option text</label>

If we are generating radiobuttons, the result looks something like this: <input type="radio" value="1" checked ...><label ...>option 1 text</label>
<input type="radio" value="2" ...><label ...>option 2 text</label>
<input type="radio" value="3" ...><label ...>option 3 text</label>

The number of lines in the result depends on the number of items in the options array in \$item. In case of a checkbox there should only be one, in case of radiobuttons there should be more than 1.

There are two different ways to specify the options. The simple way is to have a single options array with 'value' => 'option text' pairs. In this case the available properties such as

title, class and viewonly are copied from the corresponding properties in the \$item array,

The other way is to have an array of arrays like this:

```
$item['options'] = array('1'=>array('title'=>'...', 'option'=>'...'), 2 => array(...));
```

This allows for setting properties of individual options, e.g. one of the radio buttons could be made viewonly while the others are still selectable.

Note that such a non-simple array of arrays doesn't make much sense for a single, simple checkbox.

The properties recognised translate to the following HTML-code/attributes

name	: name
------	--------

value	: value AND perhaps 'checked' if value matches option value
-------	---

accesskey	: accesskey
-----------	-------------

alt	: alt
-----	-------

class	: class (also depends on viewonly and errors)
-------	---

tabindex	: tabindex
----------	------------

id	: id
----	------

title	: title
-------	---------

viewonly	: disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
----------	---

errors	: addition of ATTR_CLASS_ERROR to class list (if errors > 0)
--------	--

label	: tilde+letter may change the accesskey
-------	---

Note: In case of radiobuttons, the document-wide unique id is constructed from the specified id by appending an underscore and an indexnumber (except for the first item in the list). This id can be overruled by an id specified in the options array-of-arrays.

Even when an item has an explicit accesskey, the access key can be overruled by the 'hotkey' derived from a tilde+letter combination in the options array, either in the simple case or in case of an array-of-arrays.

Note that also the generic title can be overruled by a title that is defined in the options array-of-arrays.

array/string function `dialog_get_widget_richtextinput(&$item, $name, $value, $f_type)` [line 1095]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the (current) value of the input element to show ('field value')
- **\$f_type**

construct an input field using the user's preferred editor

this constructs an input for rich text like a page with HTML-code. Most users will probably have selected the FCKeditor. However, there is also the option to use the so-called 'plain' editor, which is nothing more than a textarea in disguise.

The properties recognised translate to the following HTML-code/attributes

name	: name
------	--------

value	: value
-------	---------

accesskey	: accesskey
-----------	-------------

rows : rows
columns : cols
maxlength : maxlength
alt : alt
class : class (also depends on viewonly and errors)
tabindex : tabindex
id : id
title : title
viewonly : disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
errors : addition of ATTR_CLASS_ERROR to class list (if errors > 0)

- **TODO** if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

array/string function `dialog_get_widget_submit(&$item, $name, $value, $f_type)` [line 764]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the button's label possibly including a tilde indicating hotkey
- *string* **\$f_type** the type of the field (eg text, number, date, time, ...)

construct a submit button

this constructs a submit button. For compatibility we use a simple input of type submit because the button widget is only available since HTML 4. We may change that in the future, and force everyone to use at least HTML 4. For now it is as it is.

Note that the label of the button is retrieved from \$value rather than from the label property. We do use the \$value as a string possibly containing hotkeys (via prepending a letter with a tilde) and we also set the accesskey to that value. However, it is different from other widgets because an input cannot display underlines (a button can).

The properties recognised translate to the following HTML-code/attributes

name	: name
value	: value
accesskey	: accesskey
alt	: alt
class	: class (also depends on viewonly and errors)

tabindex : tabindex
id : id
title : title
viewonly : disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
errors : addition of ATTR_CLASS_ERROR to class list (if errors > 0)

array/string function dialog_get_widget_textinput(&\$item, \$name, \$value, \$f_type) [line 675]

Function Parameters:

- *array* **&\$item** the parameters that describe the dialog input element
- *string* **\$name** the name of the input element ('fieldname')
- *mixed* **\$value** the (current) value of the input element to show ('field value')
- *string* **\$f_type** the type of the field (eg text, number, date, time, ...)

construct an input field, usually for text input OR a textarea for multiline input

this constructs most variations on text fields, including password fields. Many of the defined field types (the F_* constants) can be handled via a simple input of type text. The semantics of the field (eg. is it an integer, a real) have no impact on the HTML-input: at that level it is still plain text. However, for a password we use the password type in order to make the value display as asterisks. If the number of rows is more than 1, the input element becomes a text area. Note that this generally only applies to F_ALPHANUMERIC but that is not enforced here (you can make a multiline F_DATE, even though it doesn't make much sense).

The properties recognised translate to the following HTML-code/attributes

name	: name
------	--------

value	: value
-------	---------

accesskey	: accesskey
-----------	-------------

rows	: rows (textarea only)
------	------------------------

columns	: cols (textarea) or size (input type="text")
---------	---

maxlength	: maxlength
-----------	-------------

alt	: alt
-----	-------

class	: class (also depends on viewonly and errors)
-------	---

tabindex	: tabindex
----------	------------

id	: id
----	------

title	: title
-------	---------

viewonly	: disabled AND addition of ATTR_CLASS_ERROR to class list (if viewonly == TRUE)
----------	---

errors	: addition of ATTR_CLASS_ERROR to class list (if errors > 0)
--------	--

- **TODO** if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

array function dialog_quickform(\$href, &\$dialogdef, [\$method = 'post'], [\$attributes = '']) [*line 233*]

Function Parameters:

- *string* **\$href** the target of the HTML form
- *array* **&\$dialogdef** the array which describes the complete dialog
- *string* **\$method** method to submit data to the server, either 'post' or 'get'
- *string/array* **\$attributes** holds the attributes to add to the form tag

construct a generic form with a dialog

this constructs an HTML form with a simple dialog where

- every label and every widget has its own line
(enforced by a BR-tag)
- label/widget-combinations are separated with a P-tag
- buttons are stringed together on a single line (ie no trailing BR)

This should be sufficient for many dialogs. If the layout needs to be more complex a custom dialog can always be constructed using functions [dialog_get_label\(\)](#) and [dialog_get_widget\(\)](#).

- **Used by** [ConfigAssistant::show_dialog\(\)](#)
- **Uses** [html_form\(\)](#)

bool function dialog_validate(&\$dialogdef) [*line 434*]

Function Parameters:

- *array* **&\$dialogdef** the complete dialog definition; contains detailed errors and/or reformatted values

validate and check values that were submitted via a user dialog

this steps through the definition of a dialog and retrieves the values submitted by the user via `$_POST[]`. The values are checked against the constraints (e.g. minimum string length, date range, etc.). If the submitted value is considered valid, it is stored in the corresponding value of the dialogdef element, maybe properly reformatted (in case of dates/times/datetimes). If there were errors, these are recorded in the dialog definition element, in the form of one or more readable error messages. Also the error count (per element) is incremented. This makes it easy to

- inform the user about what was wrong with the input data
- determine whether there was an error at all (if `$dialogdef[$k]['errors'] > 0`).

Note that this routine has the side effect of filling the dialog array with the data that was submitted by the user via `$_POST`. If the validation is successful, the data is ready to be saved into the database. If it is not, the data entered is still available in the dialogdef which makes it easy to return to the user and let the user correct the errors without losing all the data input because of a silly mistake in some input field.

Update 2009-03-17: We no longer validate the view-only fields because these fields are not POST'ed by the browser and hence cannot be validated. This also means that there is no value set from `$_POST` for those fields.

- Used by [ConfigAssistant::save_data\(\)](#)

bool function valid_datetime(\$f_type, \$input, &\$output) [line 1408]

Function Parameters:

- *string* **\$f_type** indicates the field type we are expecting, can be F_DATE, F_TIME or F_DATETIME
- *string* **\$input** the string that needs to be checked
- *string* **&\$output** if the input is valid, this contains a properly formatted value

check validity of date, time or datetime

this checks the validity of dates and times. If all tests are passed successfully, the input value is reformatted in the standard format corresponding with that field type:

- F_DATE becomes yyyy-mm-dd (with leading zeros for month or day where applicable)
- F_TIME becomes hh:mm:ss (with leading zeros when applicable)
- F_DATETIME is combination of F_DATE and F_TIME glued together with a space: yyyy-mm-dd hh:mm:ss

Valid values for dates are within the range 0000-01-01 ... 9999-12-31 (but note that the year is always displayed with 4 digits). This routine takes leap years into account the same

way the standard function `checkdate()` does.

Valid values for times are between 00:00:00 and 23:59:59. Note that we don't deal with leap seconds or other fancy stuff (this is not rocket science): KISS. Usually we only need times to determine an embargo date/time anyway.

Also, this routine doesn't know about time zones and daylight savings time.

email.class.php

/program/lib/email.class.php - wrapper for sending mail

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: email.class.php,v 1.1.1.1 2011-02-01 13:00:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

filelib.php

/program/lib/filelib.php - utilities for manipulating files

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: filelib.php,v 1.1.1.1 2011-02-01 13:00:11 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool/string function get_mediatype(\$mimetype) [line 353]

Function Parameters:

- *string* **\$mimetype** the full mimetype to examine, possibly with parameters

extract the mediatype and -subtype from a full mimetype

this extracts the mediatype and -subtype from a full mimetype, i.e. 'text/plain' from 'text/plain; charset=US-ASCII' (see also [get_mimetype\(\)](#) and RFC2616). If \$mimetype doesn't look like a mimetype, we return FALSE.

string function get_mimetype(\$path, [\$name = ""]) [line 290]

Function Parameters:

- *string* **\$path** fully qualified path to the file to test
- *string* **\$name** name of the file, possibly different from \$path

determine the mimetype of a file

This routine tries to discover the mimetype of a file. First we try to determine the mimetype via the fileinfo extension. If that doesn't work, we try the deprecated mime_content_type() function. If that doesn't work, we try to shell out to file(1). If that doesn't work, we resort to "guessing" the mimetype based on the extension of the file or else we return the generic 'application/octet-stream'.

Note that in step 3 we shell out and try to execute the `file(1)` command. The results are checked against a pattern to assert that we are really dealing with a mime type. The pattern is described in RFC2616 (see sections 3.7 and 2.2):

```
media-type = type "/" subtype *( ";" parameter )
type       = token
subtype    = token
token       = 1*<any CHAR except CTLs or separators>
separators = "(" | ")" | "<" | ">" | "@"
           | "," | ";" | ":" | "\" | "<">
           | "/" | "[" | "]" | "?" | "="
           | "{" | "}" | SP | HT
CHAR       = <any US-ASCII character (octets 0 - 127)>
CTL        = <any US-ASCII control character
           (octets 0 - 31) and DEL (127)>
SP         = <US-ASCII SP, space (32)>
HT         = <US-ASCII HT, horizontal-tab (9)>
"<">      = <US-ASCII double-quote mark (34)>
```

This description means we should look for two tokens containing letters a-z or A-Z, digits 0-9 and these special characters: ! # \$ % & ' * + - . ^ _ ` | or ~. That's it.

Note that `file(1)` may return a mime type with additional parameters. e.g. `'text/plain; charset=US-ASCII'`. This fits the pattern, because it starts with a token, a slash and another token.

The optional parameter `$name` is used to determine the mimetype based on the extension (as a last resort), even when the current name of the file is meaningless, e.g. when uploading a file, the name of the file (from `$_FILES['file0']['tmp_name']`) is something like `'/tmp/php4r5dwfw'`, even though `$_FILES['file0']['name']` might read `'S6301234.JPG'`. If `$name` is not specified (i.e. is empty), we construct it from `$path`.

- **TODO** there is room for improvement here: the code in step 1 and step 2 is largely untested
- **Usedby** [send_file_from_datadir\(\)](#)

array function `get_mimetypes_array()` [line 73]

return an array with mimetypes keyed by file extension

This routine returns an array with 'known' combinations of (lower case) file extensions and (lowercase) mime types. This array can be used in two ways.

Example 1: find a mimetype by extension

```
$mimetypes = get_mimetypes_array();  
$mimetype = $mimetypes['jpg']; // this should yield 'image/jpeg'
```

Example 2: find an extension by mimetype

```
$mimetypes = get_mimetypes_array();  
$extension = array_search('image/jpeg',$mimetypes); // this should yield 'jpg'
```

Note that in that last example the first matching element is used. This implies that the most common extension for a certain mimetype should come first in the array, i.e. 'jpg'=>'image/jpeg' should come before 'jpeg'=>'image/jpeg'.

The list below is based on the list of mime types as distributed with the Apache webserver software.

Changes and tweaks to the list below:

application/octet-stream: default extension is an empty string "
application/postscript: default extension is ps
audio/mpeg: default extension is mp3
image/jpeg: default extension is jpg
text/plain: default extension is txt
video//quicktime: default extension is mov

NOTE

Please do not change the mapping for both the empty extension "" and the binary extension 'bin'; these extensions must map to 'application/octet-stream' because this is necessary to defeat tricks with uploading files with double extensions (as used in the File Manager).

filemanager.class.php

/program/lib/filemanager.class.php - filemanager

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: filemanager.class.php,v 1.1.1.1 2011-02-01 13:00:40 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

PARAM_FILENAME = filename_ [line 41]

This constant is used to construct the fieldname used for deleting files

PARAM_FILENAMES = filenames [line 44]

This constant is used to construct the fieldname counting the number of files to delete

PARAM_PATH = path [line 46]
PARAM_SORT = sort [line 47]
SORTBY_DATE_ASC = 3 [line 53]
SORTBY_DATE_DESC = -3 [line 54]
SORTBY_FILE_ASC = 1 [line 49]
SORTBY_FILE_DESC = -1 [line 50]
SORTBY_NONE = 0 [line 48]
SORTBY_SIZE_ASC = 2 [line 51]
SORTBY_SIZE_DESC = -2 [line 52]
TASK_ADD_DIRECTORY = mkdir [line 37]
TASK_ADD_FILE = upload [line 36]
TASK_CHANGE_DIRECTORY = cd [line 32]
TASK_LIST_DIRECTORY = ls [line 31]
TASK_PREVIEW_FILE = preview [line 33]
TASK_REMOVE_DIRECTORY = rmdir [line 35]
TASK_REMOVE_FILE = rm [line 34]
TASK_REMOVE_MULTIPLE_FILES = batchrm [line 38]
THUMBNAIL_PREFIX = zz_thumb_ [line 57]

This constant is used to specify thumbnail files to be ignored in directory listings

require_once \$CFG->progdir."/lib/filelib.php" [line 28]

utility routines for manipulating files

groupmanager.class.php

/program/lib/groupmanager.class.php - taking care of group management

This file defines a class for dealing with groups.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: groupmanager.class.php,v 1.1.1.1 2011-02-01 13:00:23 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

GROUPMANAGER_MAX_CAPACITIES = 8 *[line 30]*

this defines the maximum number of capacities a group can have (keep this below 10 because of dialog hotkeys)

htmllib.php

/program/lib/htmllib.php - useful functions for generating HTML-code

This file provides various utility routines that aid in creating HTML-code.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmllib.php,v 1.1.1.1 2011-02-01 13:00:13 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

void function href(\$path, [\$params = "], [\$fragment = "]) [line 204]

Function Parameters:

- *string* **\$path** the hypertext reference
- *array|string* **\$params** the parameter(s) to add to the \$path
- *string* **\$fragment** the optional position within the page

construct a href from a path, params and a fragment

- **TODO** should we merge this with `html_a()` and/or rename this routine to `html_href()`?

string function html_a([\$href = "], [\$params = NULL], [\$attributes = NULL], [\$anchor = NULL]) [line 69]

Function Parameters:

- *string* **\$href** holds the hypertext reference
- *string|array* **\$params** holds the parameters to add to the \$href
- *string|array* **\$attributes** holds the attributes to add to the tag

- *string* **\$anchor** if not empty this string and a closing tag are appended

construct an HTML A tag with optional parameters and attributes

this constructs an A tag of the form ``

If no parameters are specified, nothing is added to the href. If no attributes are specified the tag only has the href attribute. If **\$params** is an array, the elements of this array are added to the href after a `rawurlencode()`. The complete **\$href** is then escaped via `htmlspecialchars()`. If **\$attributes** is an array, all elements are added as escaped key-value-pairs. If string, then just append. If **\$anchor** is not empty, the string is appended to the constructed opening tag and subsequently a closing tag is appended.

Note that urlencoding and specialchars are applied to the URL property and that the other properties are only are `htmlspecialchars()`ed. The optional **\$anchor** is not changed in any way.

Examples: `html_a('index.php')`: ``
`html_a('index.php',array('foo'=>'bar'))`: ``
`html_a('index.php',array('x'=>'y'),array('title'=>'foo'))`: ``
`html_a('index.php',"",array('class'=>'dimmed'),'baz')`: `baz`
`html_a("","",array('name'=>'chapter1'),'chapter 1')`: `chapter 1`

string function `html_attributes($attributes)` [line 159]

Function Parameters:

- *mixed* **\$attributes** an array or string with attributes or NULL

convert an array of name-value pairs to a string

this converts an array of name-value-pairs to a string containing `attribute="content"` items, where both 'attribute' and 'content' are properly escaped (with `htmlspecialchars()`). Properties that don't have content, such as 'disabled' or 'selected' or 'checked' can be specified using the special value NULL, e.g. `array('disabled' => NULL)`. If the parameter **\$attributes** happens to be a string, it is returned with a space prepended. If it is neither a string nor an array (e.g. NULL), an empty string is returned. Note that the `attribute="content"` elements are delimited with spaces, and that a leading space is prepended (but not trailing space is added).

string function `html_form($action, [$method = 'post'], [$attributes = ''])` [line 182]

Function Parameters:

- *string* **\$action** the url to submit to
- *string* **\$method** either get or post (default)
- *string|array* **\$attributes** holds the attributes to add to the tag

construct the opening of a HTML form

- Usedby [dialog_quickform\(\)](#)
- Usedby [TranslateTool::render_translation_dialog\(\)](#)

void function `html_form_close()` [*line 191*]

companion of `html_form`: close the tag

string function `html_img([$src = "], [$attributes = NULL])` [*line 115*]

Function Parameters:

- *string* **\$src** holds the url to the image file
- *string|array* **\$attributes** holds the attributes to add to the tag

construct an HTML IMG tag with optional attributes

this constructs an IMG tag of the form ``

If no attributes are specified the tag only has the src attribute If \$attributes is an array, all elements are added as raw encoded key-value-pairs. If it is a string, then just append.

Examples: `html_img('icon.gif')`: ``

`html_img('icon.gif',array('width'=>16, 'height'=>16))`: ``

void function `html_input_radio($name, $options)` [*line 241*]

Function Parameters:

- **\$name**
- **\$options**

STUB

void function `html_input_select($name, $options)` [*line 231*]

Function Parameters:

- **\$name**
- **\$options**

STUB

void function `html_input_submit($name, $value)` [*line 250*]

Function Parameters:

- **\$name**
- **\$value**

STUB

void function `html_input_text($name)` [*line 225*]

Function Parameters:

- **\$name**

STUB

string function `html_table([$attributes = NULL], [$content = NULL], $m)` [*line 261*]

Function Parameters:

- *string/array* **\$attributes** holds the attributes to add to the tag
- *string* **\$m** margin for improved code readability

- **\$content**

construct the opening of a HTML table

void function `html_table_cell`([\$attributes = NULL], [\$content = NULL]) [*line 286*]

Function Parameters:

- **\$attributes**
- **\$content**

void function `html_table_cell_close`() [*line 291*]

string function `html_table_close`([\$m = margin for improved code readability]) [*line 271*]

Function Parameters:

- *string* **\$m** margin for improved code readability

construct table closing tag

void function `html_table_head`([\$attributes = NULL], [\$content = NULL]) [*line 296*]

Function Parameters:

- **\$attributes**
- **\$content**

void function `html_table_head_close`() [*line 301*]

void function `html_table_row`([\$attributes = NULL], [\$content = NULL]) [*line 276*]

Function Parameters:

- **\$attributes**
- **\$content**

void function `html_table_row_close`() [*line 281*]

string function `html_tag`[\$tag = "], [\$attributes = NULL], [\$content = NULL]) [*line 133*]

Function Parameters:

- *string* **\$tag** is the HTML-tag to create, e.g. 'span' or 'script'
- *mixed* **\$attributes** holds the attributes to add to the tag or NULL
- *mixed* **\$content** if not NULL this string and a closing tag are appended

construct a generic HTML-tag with attributes, optionally close it too

language.class.php

/program/lib/language.class.php - taking care of translations of messages

This file defines a class for dealing with translation of phrases.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: language.class.php,v 1.1.1.1 2011-02-01 13:00:17 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

loginlib.php

/program/lib/loginlib.php -- functions to handle user login/logout

Visitors need to authenticate when they want to see a 'protected' area or when they want to modify the website content. This requires a user account and the visitor presenting valid credentials (username + password).

We don't want malicious scripts trying to get in with brute force. However, we need to accomodate users that make typo's while entering credentials. Also we want to allow for sending password reminders, in a safe way.

Features:

- users are allowed N login attempts within an interval of T1 minutes
 - users can request a new password (a 'bypass') to be mailed to them.
this additional password is valid for only T2 minutes
 - if a user has requested a bypass, the user is forced to change her password. the new password must differ from the old password and also from the bypass
 - if too many failures are detected in the last T1 minutes, login attempts from the corresponding IP-address are blocked for T3 minutes
- N = \$CFG->login_max_failures, default 10

T1 = \$CFG->login_failures_interval, default 12 minutes

T2 = \$CFG->login_bypass_interval, default 30 minutes

T3 = \$CFG->login_blacklist_interval, default 8 minutes

Once a user is authenticated, a PHP-session is established, using our own database based session handler. The session key is stored in a cookie in the user's browser. Presenting this cookie on subsequent calls is enough to gain access. The logout routine takes care of killing both the user's cookie and the session in the database.

There are several different login procedures.

1. Normal login

The user enters a valid username and password and is subsequently logged in.

2. Change password The user enters a valid username and password and also a valid new password (twice). A salted hash of the new password is recorded in the database and the user is logged in.

3. Forgotten password, phase 1: sending a laissez-passer The user presents a valid combination of username and email address. Subsequently a one-time logon-code (dubbed 'laissez-passer') is sent to the user's email address. This code is valid for at most T2 minutes. This code can be used, exactly once, to send a temporary password via email.

4. Forgotten password, phase 2: sending a temporary password The user clicks the link

received in phase 1 and a temporary password (dubbed 'bypass') is sent to the user. This temporary password is valid for another T2 minutes.

5. Message box This is a pseudo-procedure. A simple 'message box' type of screen is displayed but no real interaction is anticipated via this screen. This is used to tell the user that things didnt work out (too many failures) or to check their mail for further instructions (e.g. when a laissez passer was sent). Whenever the user acknowledges this screen by clicking the button, she usually is directed to \$WAS_SCRIPT_NAME.

6. Blacklist This is also a pseudo-procedure. The corresponding number is used to identify blacklisted IP-addresses in the database.

Note that when the user logs in after a temporary password has been sent, the normal login procedure is immediately followed by a (forced) 'change password' procedure. This makes sure that a temporary password will be changed immediately after the user logs in.

Note that each of the procedures can be entered 'manually', i.e. by opening index.php?login=X the user starts procedure X. This allows for the user to change her password whenever she feels this is necessary, without going through the trouble of the 'forgotten password'-procedure which eventually ends with the user changing her password too.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: loginlib.php,v 1.1.1.1 2011-02-01 13:00:48 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** should we suppress the usernamd in the laissez-passer routine? We _do_ leak the the username in an insecure email message. This does require making the laissez-passer code unique in the database (currently only username+code has to be unique and that's easy because the username itself is unique).
- **Usedby** [admin_show_login_and_exit\(\)](#)
- **Usedby** [admin_login\(\)](#)
- **Usedby** [admin_logout_and_exit\(\)](#)
- **License** [GNU AGPLv3+Additional Terms](#)

BY_EMAIL = 2 [line 153]

this selects authentication via username+email in authenticate_user()

BY_LAISSEZ_PASSER = 3 *[line 156]*

this selects authentication via username+laissez_passer in authenticate_user()

BY_PASSWORD = 1 *[line 150]*

this selects authentication via username+password in authenticate_user()

LOGIN_DEBUG = 0 *[line 111]*

useful when debugging routines in this file: 0=production, 1=debugging

LOGIN_FAILURE_DELAY_SECONDS = 3 *[line 135]*

this is the number of seconds to delay responding after a login action fails (slow 'm down..)

LOGIN_PROCEDURE_BLACKLIST = 6 *[line 132]*

this is a pseudo procedure, used to record blacklisted IP-addresses

LOGIN_PROCEDURE_CHANGE_PASSWORD = 2 *[line 120]*

this is the procedure to change the user's password

LOGIN_PROCEDURE_MESSAGE_BOX = 5 *[line 129]*

this is a pseudo procedure, used to deliver some message to the user

LOGIN_PROCEDURE_NORMAL = 1 *[line 117]*

this is the usual procedure for logging in

LOGIN_PROCEDURE_SEND_BYPASS = 4 *[line 126]*

this is phase 2 of the 'forgot password' procedure

LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER = 3 *[line 123]*

this is phase 1 of the 'forgot password' procedure

LOGIN_PROCEDURE_SHOWLOGIN = 0 *[line 114]*

this only shows the login dialog

MAXIMUM_LINE_LENGTH = 50 *[line 159]*

this defines the maximum line length in messages and instructions

MINIMUM_PASSWORD_DIGITS = 1 *[line 147]*

this is the hardcoded minimal number of digits in a new password

`MINIMUM_PASSWORD_LENGTH = 6 [line 138]`

this hardcoded minimal length is enforced whenever a user wants to change her password

`MINIMUM_PASSWORD_LOWERCASE = 1 [line 141]`

this is the hardcoded minimal number of lower case characters in a new password

`MINIMUM_PASSWORD_UPPERCASE = 1 [line 144]`

this is the hardcoded minimal number of upper case characters in a new password

`bool function acceptable_new_password($new_password1, $new_password2, [$salt = "], [$password_hash = "],
[$bypass_hash = "]) [line 1272]`

Function Parameters:

- *string* **\$new_password1** new password
- *string* **\$new_password2** new password again, to prevent typo's
- *string* **\$salt** (optional) the salt that was used to hash the old password and the bypass
- *string* **\$password_hash** (optional) the hashed existing password
- *string* **\$bypass_hash** (optional) the hashed bypass password

check the new passwords satisfy password requirements

Users should provide the same password twice, to prevent typo's, so both passwords should be equal. Also, the following requirements should be satisfied:

- the minimum password length should be `MINIMUM_PASSWORD_LENGTH` (default 6)
- the new password should contain at least `MINIMUM_PASSWORD_LOWERCASE` lowercase letter a-z (default 1)
- the new password should contain at least `MINIMUM_PASSWORD_UPPERCASE` upper case letter A-Z (default 1)
- the new password should contain at least `MINIMUM_PASSWORD_DIGITS` digit 0-9 (default 1)
- the new password should not be the same as the previous password
- the new password should not be the same as the bypass password (if any was issued)

Note that the bypass-generator also should satisfy these rules. This could lead to the thought of accepting the bypass-password as the permanent one. However, since this temporary password was sent to the user in a plain-text email message, we should consider this a 'bad' password.

The minimum password length and other minimum values are not configurable (via `$CFG`)

because that would make it too easy (too tempting) to give in and use weak passwords (too short, only lowercase, etc.) However, if you really **MUST**, you could change the `MINIMUM_PASSWORD_*` constants defined above.

Note that the check against existing (temporary and regular) passwords is not performed if the corresponding parameters are empty. If they are empty, this routine only performs the first 4 checks in the list above.

bool/array function `authenticate_user($by_what_token, $username, $token)` [line 1168]

Function Parameters:

- *int* **\$by_what_token** which authentication token to use
- *string* **\$username** username the user entered in the dialog
- *string* **\$token** the token is either password, email or laissez_passer entered by the user

check the user's credentials in one of three ways

This authenticates the user's credentials. There are some variants:

- by password: the user's password should match
- by email: the user's email should match
- by laissez passer: the one-time authentication code should match

Strategy: we first read the active record for user `$username` in core. If there is none, the user does not exist or is inactive => return FALSE.

After that we check the validity of the token:

- a password is checked via the password hash or, if that fails, via the bypass hash. In the latter case, the bypass should not yet be expired (a bypass and a laissez_passer are valid until the 'bypass_expiry' time).
- an email address is checked caseInsensitive and without leading/trailing spaces
- a laissez_passer is checked much the same way as the bypass password, but that the code is stored 'as-is' rather than as a hash. The comparison is caseInsensitive.

If the credentials are considered valid, an array with the user record is returned, otherwise FALSE is returned.

Because there are actually several checks to be done, we decided not to use SQL like: `SELECT * FROM users WHERE username=$username AND password=$password`, not the least because we need to have the salt in our hands before we can successfully compare password hashes.

Note: The 'special cases' (checking email, checking laissez_passer, checking bypass) all have their token stripped from leading and trailing spaces. We don't want to further confuse the user by not accepting a spurious space that was entered in the heat of the moment when the user has 'lost' her password. Therefore we also always trim the username. Rationale: usernames and also the generated passwords etc. never have leading/trailing spaces. However, one cannot be sure that a user has not entered a real password with leading/trailing space, so we do NOT trim the \$token in the first attempt in the case 'BY_PASSWORD' below.

bool function login_change_password(\$user_id, \$new_password) [line 1221]

Function Parameters:

- *int* **\$user_id** identify the user record by user_id
- *string* **\$new_password** the new password in plain text

update the users database with a new (randomly salted) password and reset bypass mode to normal

This updates the user record for user with user_id and stores the new password. The new password and a new random salt are hashed together and the result is stored, together with the new salt, overwriting the old salt and the old password hash. The bypass mode is reset to normal and the bypass hash is reset. Return TRUE on success.

string function login_dialog_close([\$action = "], [\$m = "]) [line 840]

Function Parameters:

- *string* **\$action** (optional) if not empty the currently open HTML-form is closed
- *string* **\$m** (optional) margin to add for code readability

close the login dialog/table and maybe an opened HTML-form

- See [login_dialog_open\(\)](#)

string function login_dialog_home_forgot_password(\$forgot, [\$m = "]) [line 936]

Function Parameters:

- *string* **\$forgot** anchor text to display with link to forgot password dialog
- *string* **\$m** (optional) margin to add for code readability

add a row with links to home page and forgot password dialog to the login dialog/table

This constructs a link to the home page in the left hand dialog/table column and optionally a link to the start of the forgot password dialog. The latter is only displayed if \$forgot is not empty.

string function login_dialog_instruction(\$instruction, [\$m = ""]) [*line 852*]
Function Parameters:

- *string* **\$instruction** instructive message to show to user
- *string* **\$m** (optional) margin to add for code readability

add a row to the table/dialog with wordwrap()'ed instruction for the user

string function login_dialog_open(\$title, [\$action = ""], [\$message = ""], [\$m = ""]) [*line 796*]
Function Parameters:

- *string* **\$title** text to show in the dialog title bar
- *string* **\$action** (optional) if not empty a HTML-form pointing to \$action is opened
- *string* **\$message** (optional) feedback message to show to user
- *string* **\$m** (optional) margin to add for code readability

construct the start of the login dialog, opening the form and the secondary table

This optionally opens an HTML-form (which is optionally closed in companion routine [login_dialog_close\(\)](#)) and subsequently starts a table with two columns. The first row of the table shows the \$title, the second row may show a wordwrap()'ed feedback message for the user in a different background colour.

- See [login_dialog_close\(\)](#)

- **TODO** should we add another 'powered by' link to '/program/about.html'?

string function login_dialog_password_input(\$prompt, \$name, [\$tabindex = "], [\$m = "]) [*line 893*]

Function Parameters:

- *string* **\$prompt** the text to show in the 1st column of the table row
- *string* **\$name** the name of the input field
- *int* **\$tabindex** (optional) determines in which order fields are accessed in the dialog
- *string* **\$m** (optional) margin to add for code readability

add a row with a password input field to the login dialog/table

This generates HTML for another table row. Special feature: try to suppress autocomplete via an extra parameter. See

string function login_dialog_submit_input(\$buttontext, \$name, [\$tabindex = "], [\$m = "]) [*line 913*]

Function Parameters:

- *string* **\$buttontext** the text to show in the button
- *string* **\$name** the name of the button
- *int* **\$tabindex** (optional) determines in which order fields are accessed in the dialog
- *string* **\$m** (optional) margin to add for code readability

add a row with a submit button to the login dialog/table

string function login_dialog_text_input(\$prompt, \$name, [\$value = "], [\$tabindex = "], [\$m = "]) [*line 869*]

Function Parameters:

- *string* **\$prompt** the text to show in the 1st column of the table row
- *string* **\$name** the name of the input field
- *string* **\$value** (optional) value to preload the field value (default empty string)
- *int* **\$tabindex** (optional) determines in which order fields are accessed in the dialog

- *string* **\$m** (optional) margin to add for code readability

add a row with an ordinary input field to the login dialog/table

bool/int function login_failure_blacklist_address(\$remote_addr, \$delay_in_seconds, [\$username = "]) [*line 1426*]

Function Parameters:

- *string* **\$remote_addr** the remote IP-address is the origin of the failure
- *int* **\$delay_in_seconds** the number of seconds to put this address on the blacklist
- *string* **\$username** extra information, could be useful for troubleshooting afterwards

add remote_addr to the blacklist for specified interval (in seconds)

bool/int function login_failure_delay(\$remote_addr) [*line 1460*]

Function Parameters:

- *string* **\$remote_addr** the remote IP-address that is the origin of the failure

delay execution of this script for a few seconds and blacklist the remote_addr during the delay

This immediately blacklists the remote address for LOGIN_FAILURE_DELAY_SECONDS seconds. Once that is done, the execution is delayed for that same period of time. After the delay, the temporary blacklisting is removed from the table. The whole purpose of this rapid succession of an INSERT and a DELETE is to prevent brute force attack scripts that do not wait for an answer and/or use multiple connections. This routine defeats that trick, because nothing can be done when an IP-address is blacklisted.

- **Uses \$CFG**

int function login_failure_increment(\$remote_addr, \$procedure, [\$username = "]) [*line 1481*]

Function Parameters:

- *string* **\$remote_addr** the remote IP-address that is the origin of the failure
- *int* **\$procedure** indicates in which procedure the user failed
- *string* **\$username** extra information, could be useful for troubleshooting afterwards

add 1 point to score for a particular IP-address and failed procedure, return the new score

This records a login failure in a table and returns the the number of failures for the specified procedure in the past T1 minutes.

bool function login_failure_reset(\$remote_addr) [line 1415]

Function Parameters:

- *string* **\$remote_addr** the remote IP-address is the origin of the failure

deactivate all login failures/blacklisting scores for remote_addr

This resets all the scores for all failed login attempts and blacklistings for the specified IP-address. The records in the login_failures table are deactivated by deleting the records for this remote_addr.

Note that the failed logins and the blacklistings are recorede in the log_messages table via [logger\(\)](#). Therefore we can automatically keep this table 'login_failures' clean without cron jobs.

This routine resets **_all_** scores, including any blacklisting that might still be active, i.e. which has a datim in the future.

bool function login_is_blacklisted(\$remote_addr) [line 1384]

Function Parameters:

- *string* **\$remote_addr** the remote IP-address to be checked

find out if a remote address is blacklisted at this time

This routine checks if this remote address is blacklisted in the login_failures table with a datim that lies in the future. If this is the case, the address is indeed blacklisted and TRUE is returned. Note that we sum the points much the same way as in [login failure increment](#) rather than counting 'blacklist-records'.

string function login_page_close([\$alert_message = ""]) [*line 763*]

Function Parameters:

- *string* **\$alert_message** (optional) message to show via a javascript alert()

construct the end of the simple HTML-page, closing the full size table

- See [login_page_open\(\)](#)
- Uses [javascript_alert\(\)](#)

string function login_page_open(\$title, [\$focus = ""]) [*line 699*]

Function Parameters:

- *string* **\$title** the title of the HTML-page
- *string* **\$focus** (optional) the name of the field to focus on (via Javascript)

construct the start of a simple HTML-page and open a full size table

This routine starts with a plain HTML-page, with a title and possibly a single line of Javascript to place the cursor in a particular input field (defined later in the page). After that, a main table is opened and within that 1x1 table the table cell is opened. [login_page_close\(\)](#) closes that cell.

- See [login_page_close\(\)](#)
- Uses \$CFG;

bool function login_send_bypass(\$user) [*line 1046*]

Function Parameters:

- **array \$user** an associative array with the user record

send a new (temporary) password to the user via email

This generates a new temporary password for the user, stores it in the user record and sends an email message to the user with the temporary password (in plain text) and further instructions.

Note that the password is valid only for a limited time; sending a password in plain text appears to be an acceptable risk. Note that the limited time is increased with 10% in order to give the user a reasonable margin to enter the correct password.

Also note that the existing salt is used to salt the temporary password; this makes it easier to check for validity of both the regular password and the temporary password later on.

A log message recording the event is added via [logger\(\)](#).

- **Uses** [logger\(\)](#)
- **Uses** \$CFG

bool function login_send_confirmation(\$user) [*line 1100*]

Function Parameters:

- **array \$user** an associative array with the user record

send email to user confirming password change

This sends an email to the user's email address confirming that the user's password was changed. Note that the new password is NOT sent to the user.

- **Uses** \$CFG

bool function login_send_laissez_passer(\$user) [*line 980*]

Function Parameters:

- *array* **\$user** the user record from database

send a special one-time login code to the user via email

This generates a temporary code with which the user can request a new temporary password. This code can be used only once. Note that this code is valid for only a limited time. This code simply overwrites the bypass password (the temporary password) in the user record. This means that if a phase 2 is pending, a new phase 1 will replace the old phase 2.

The temporary code consists of digits and uppercase characters. However, it is longer (20 characters) than the minimum password length of 6, so a brute force on such a code will likely not succeed (36^{20} is much more than the usual 62^6).

This routine also brings the user's record into 'bypass mode'. This mode is reset to 'normal' after the user has successfully changed her password.

A log message recording the event is added via [logger\(\)](#).

- **Uses** [logger\(\)](#)
- **Uses** \$CFG

string function login_stylesheet() [*line 724*]

a simple in-line style sheet conveniently grouped in a single routine

- **TODO** this routine needs some cleaning up

string function password_hash(\$salt, \$password, [\$algorithm = 0]) [*line 1326*]

Function Parameters:

- *string* **\$salt**
- *string* **\$password**

- *int* **\$algorithm** (optional) algorithm to use: 0=md5, 1=sha1

calculate a hash from a salt and a password

This routine constructs a hash of the combination of salt and password. By default the md5() function is used to calculate a 32-character long string of hexadecimal digits. If the parameter \$algorithm is 1 then the sha1() function is used and a 40-character long string of hexadecimal digits is returned.

Note that we do not use the crypt() function because that could introduce a portability issue. If a website is migrated to another machine, the used crypt algorithm might no longer be available, and that would effectively lock out all users. Both md5() and sha1() are standard PHP-functions (since 4.3.x) and should be portable, which makes any installed table of users portable too.

- Used by [password_hash_check\(\)](#)

bool function password_hash_check(\$salt, \$password, \$hash) [*line 1353*]

Function Parameters:

- *string* **\$salt** salt
- *string* **\$password** password to check
- *string* **\$hash** hash to check against

check equivalency of salt+password against hash

This verifies whether the hash of \$salt and \$password is the same as \$hash. Note that the two hashes are compared in a caseInsensitive way. Usually these hashes are using lowercase hexadecimal digits but a caseInsensitive compare makes A,...,F equivalent to a,...,f.

If the length of the presented \$hash is 40 characters, it is assumed that the hash algorithm to use is sha1, otherwise the default algorithm (md5) is used.

- Uses [password_hash\(\)](#)

string function password_salt([\$length = 12]) [*line 1367*]

Function Parameters:

- *int* **\$length** the number of characters in the generated string

generate a quasi random string to salt the password hash

this generates a quasi-random string of digits and letters to be used as a salt when calculating a password hash.

void function show_login([\$screen = 1], [\$message = "], [\$username = "]) [*line 573*]

Function Parameters:

- *int* **\$screen** the screen variant to show, could be 1,...,5
- *string* **\$message** the message to show just above the first field, used for feedback to user
- *string* **\$username** the default username to show in the dialog

show complete login dialog and exit

There are different variations of this dialog.

1. LOGIN_PROCEDURE_NORMAL Plain login

(message)

Username: _____

Password: _____

[OK]

<home page> <forgotten password?>

This screen is used for plain user authentication. As a rule the user uses the correct primary password to authenticate. However, it is also possible to enter the 'bypass' password instead. If the authentication fails, that fact is recorded. If the number of failures exceeds threshold N, the user is shown screen #3 LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER. If the number is still below N1 screen #1 is shown again.

The link <forgotten password?> takes the user directly to screen #3 LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER.

2. LOGIN_PROCEDURE_CHANGE_PASSWORD - Login/change password (message)

Username: _____
Old password: _____
New password1: _____
New password2: _____

[OK] This screen is used to change the user's password. If both new passwords are different, the user is redirected to the same screen #3 until she gets it right. Otherwise, if the old password is either the valid original password OR the bypass password, the password is changed and the mode is reset to 'normal'. The one-time codes and the bypass password are reset. Also, as a result, the user is logged in. If the user failed to enter the proper old password more than N1 times, the mode is also reset to normal (invalidating the laissez-passer and the bypass password) and the user is dropped at a screen #4 basically telling her to contact the webmaster. In this process the user is also logged out if necessary.

3. LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER Request bypass (message)

Username: _____
Email: _____

[OK] This screen is used to help the user reset her password. It is displayed automatically after N1 failed login attempts. This screen can also be reached via the <forgot password?> link in screen #1.

If the user presents an invalid combination of username and email address, this failure is also recorded. If the number of failures has reached the threshold N2, the user is taken to a screen #4 that basically tells the user ask the webmaster for assistance and that's that.

If the user presents a valid combination of username and email address, an email with a message like 'click the link below for a new password' is sent to the email address. After that mail is sent a screen #4 is displayed, basically telling the user to await further instructions that were sent via mail.

Note that resetting the password is a two-step process. First the user is sent a one-time code laissez-passer embedded in a link. Clicking the link before it expires (after T minutes) yields a second email message containing a bypass password that can be used to login and subsequently change the primary password. After that both the laissez-passer and the bypass password are invalidated.

4. LOGIN_PROCEDURE_SEND_BYPASS - Send a temporary password

Phase 2 of the forgot password procedure.

5. LOGIN_PROCEDURE_MESSAGE_BOX Alert (message)

This screen is user to communicate various messages to the user, e.g. 'check your mail for instructions', 'contact webmaster', etc.

void/int function was_login([\$procedure = LOGIN_PROCEDURE_SHOWLOGIN], [\$message = "]) [line 261]

Function Parameters:

- *int* **\$procedure** the login procedure to execute
- *string* **\$message** the message to display when showing the login dialog

execute the selected login procedure

The login process is controlled via the parameter 'login' provided by the user via 'index.php?login=N or via the 'action' property in a HTML-form. These numbers correspond to the LOGIN_PROCEDURE_* constants defined near the top of this file. Here's a reminder:

1. LOGIN_PROCEDURE_NORMAL this is the usual procedure for logging in
2. LOGIN_PROCEDURE_CHANGE_PASSWORD this is the procedure to change the user's password
3. LOGIN_PROCEDURE_SEND_LAISSEZ_PASSER this is phase 1 of the 'forgot password' procedure
4. LOGIN_PROCEDURE_SEND_BYPASS this is phase 2 of the 'forgot password' procedure

Note that this routine only returns to the caller after either a succesful regular login (i.e. after completing LOGIN_PROCEDURE_NORMAL). All the other variants and error conditions yield another screen and an immediate exit and hence no return to caller. If this routine returns, it returns the user_id of the authenticated user (the primary key into the users table). It is up to the caller to retrieve additional information about this user; any information read from the database during login is discarded. This prevents password hashes still lying around.

Note that a successful login has the side effect of garbage collection: whenever we experience a successful login any obsolete sessions are removed. This makes sure that locked records eventually will be unlocked, once the corresponding session no longer exists. The garbage collection routine is also called from the PHP session handler every once in a while, but here we make 100% sure that garbage is collected at least at every login. (Note: obsolete sessions should not be a problem for visitors that are not logged in, because you have to be logged in to be able to lock a record.)

- Uses [dbsession_setup\(\)](#)
- Uses [dbsession_garbage_collection\(\)](#)
- Uses \$CFG

void function was_logout() [*line 181*]

end a session (logout the user) and maybe redirect

This routine ends the current session if it exists (as indicated by the cookie presented by

the user's browser). An empty value is sent to the browser (effectively deleting the cookie) and also the session is ended. The routine ends either with showing a generic login dialog OR a redirection to a user-defined page.

Note that as a rule this routine does NOT return but instead calls `exit()`. However, there are cases where this routine DOES return, notably when no session appears to be established (no cookie submitted by the browser or a non-existing/expired session). If the routine does return, the status is equivalent to a logged out user; no session exists so the user simply should not be logged in.

- **Uses** [dbsession_setup\(\)](#)
- **Uses** `$CFG`

module.class.php

/program/lib/module.class.php - taking care of modules

This file defines a base class for dealing with modules. It is always included and it can be used as a base to inherit from.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: module.class.php,v 1.1.1.1 2011-02-01 13:00:10 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** we probably need to get rid of this file because it is not used (2010-12-07/PF)
- **License** [GNU AGPLv3+Additional Terms](#)

modulelib.php

/program/lib/modulelib.php - module factory

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: modulelib.php,v 1.1.1.1 2011-02-01 13:00:24 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** we probably need to get rid of this file because it is not used (2010-12-07/PF)
- **License** [GNU AGPLv3+Additional Terms](#)

bool/object function module_factory([\$module_id = 0], [\$node_id = NULL]) [*line 46*]

Function Parameters:

- *int* **\$module_id** which module to retrieve from database via primary key
- **\$node_id**

manufacture a module object

This loads (includes) a specific module based on the parameter \$module_id. Relevant data is read from the database. If no module can be found, the function returns FALSE;

Note that the base Module-class is always included so it is there if other modules need it.

- **TODO** what if the module is not found? Currently no alternative is loaded but FALSE is returned.
- **Uses** \$CFG

require_once **\$CFG->progdire**."/lib/module.class.php" [*line 30*]

module class is used as a base class from which others are derived

modulemanagerlib.php

/program/lib/modulemanagerlib.php - modulemanager

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: modulemanagerlib.php,v 1.1.1.1 2011-02-01 13:00:48 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

void function job_modulemanager(&\$output) [*line 32*]

Function Parameters:

- *object* **&\$output** collects the html output

main entry point for modulemanager (called from admin.php)

pagemanager.class.php

/program/lib/pagemanager.class.php - pagemanager

This file contains the Page Manager class, the core functionality of Website@School.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: pagemanager.class.php,v 1.1.1.1 2011-02-01 13:00:32 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

DIALOG_NODE_ADD = 1 *[line 54]*
DIALOG_NODE_DELETE_CONFIRM = 5 *[line 58]*
DIALOG_NODE_EDIT = 2 *[line 55]*
DIALOG_NODE_EDIT_ADVANCED = 3 *[line 56]*
DIALOG_NODE_EDIT_CONTENT = 4 *[line 57]*
MODULE_NAME_DEFAULT = htmlpage *[line 69]*

Default initial module of a new page (see get_dialogdef_add_node())

NODE_VISIBILIY_DEFAULT = NODE_VISIBILIY_HIDDEN *[line 67]*

Default initial visibility of a new node (see get_dialogdef_add_node())

NODE_VISIBILIY_EMBARGO = 3 *[line 65]*

Initial visibility of a new node: under embargo

NODE_VISIBILIY_HIDDEN = 2 *[line 63]*

Initial visibility of a new node: hidden

NODE_VISIBILIY_VISIBLE = 1 *[line 61]*

Initial visibility of a new node: visible

PARAM_TREEVIEW = treeview *[line 48]*
TASK_ADD_PAGE = addpage *[line 35]*
TASK_ADD_SECTION = addsection *[line 36]*
TASK_NODE_DELETE = delete *[line 37]*
TASK_NODE_EDIT = edit *[line 38]*
TASK_NODE_EDIT_ADVANCED = editadvanced *[line 40]*

TASK_NODE_EDIT_CONTENT = editcontent [line 39]
TASK_PAGE_PREVIEW = preview [line 41]
TASK_SAVE_CONTENT = savecontent [line 45]
TASK_SAVE_NEWPAGE = savenewpage [line 43]
TASK_SAVE_NEWSECTION = savenewsection [line 44]
TASK_SAVE_NODE = savenode [line 42]
TASK_SET_DEFAULT = setdefault [line 34]
TASK_SUBTREE_COLLAPSE = collapse [line 33]
TASK_SUBTREE_EXPAND = expand [line 32]
TASK_TREEVIEW = treeview [line 30]
TASK_TREEVIEW_SET = settreeview [line 31]
TREE_VIEW_CUSTOM = 2 [line 50]
TREE_VIEW_MAXIMAL = 3 [line 51]
TREE_VIEW_MINIMAL = 1 [line 49]

statisticslib.php

/program/lib/statisticslib.php - statistics

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: statisticslib.php,v 1.1.1.1 2011-02-01 13:00:23 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

void function job_statistics(&\$output) [*line 32*]

Function Parameters:

- *object* **&\$output** collects the html output

main entry point for statistics (called from admin.php)

theme.class.php

/program/lib/theme.class.php - taking care of themes

This file defines a base class for dealing with themes. It is always included and it can be used as a starting point for other themes by inheriting from this class.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: theme.class.php,v 1.1.1.1 2011-02-01 13:00:18 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

themelib.php

/program/lib/themelib.php - theme factory

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: themelib.php,v 1.1.1.1 2011-02-01 13:00:23 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool/object function theme_factory(\$theme_id, \$area_id, \$node_id) [*line 45*]

Function Parameters:

- *int* **\$theme_id** denotes which theme to retrieve from database via primary key
- *int* **\$area_id** the area we're working in
- *int* **\$node_id** the node that is to be displayed

manufacture a theme object

This loads (includes) a specific theme based on the parameter \$theme_id. Relevant data is read from the database.

- **TODO** what if the theme is not found? Currently no alternative is loaded but FALSE is returned.
- **TODO** should we massage the directory and file names of the included theme?
- **Uses** \$CFG

require_once **\$CFG->progrdir."/lib/theme.class.php"** [*line 29*]

theme class is used as a base class from which others can be derived

toolslib.php

/program/lib/toolslib.php - tools

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: toolslib.php,v 1.1.1.1 2011-02-01 13:00:50 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

TASK_BACKUPTOOL = backuptool [line 31]
TASK_LOGVIEW = logview [line 32]
TASK_TOOLS_INTRO = intro [line 29]
TASK_TRANSLATETOOL = translatetool [line 30]
void function job_tools(&\$output) [line 45]

Function Parameters:

- *object* **&\$output** collects the html output

main entry point for tools (called from /program/main_admin.php)

this routine dispatches the tasks, If the specified task is not recognised, the default task TASK_TOOLS_INTRO is executed.

- **TODO** fix permissions for backup tool! perhaps another bit?

void function show_tools_intro(&\$output) [line 113]

Function Parameters:

- *object* **&\$output** collects the html output

display an introductory text for tools + menu

void function show_tools_menu(&\$output, [\$current_task = NULL]) *[line 125]*

Function Parameters:

- *object* **&\$output** collects the html output
- *string* **\$current_task** indicate the current menu selection (if any)

display the tools menu

output function task_backuptool(&\$output) *[line 203]*

Function Parameters:

- *object* **&\$output** collects output to show to user

show an introductory text for backup tool OR stream a ZIP-file to the browser

If we arrive here via the tools menu, the parameter `download` is not set. In that case we show an introductory text with a link that yields a ZIP-file with the backup.

If the user follows the download link, we arrive here too but with the `download` parameter set. We then dump the database in a variable and subsequently compress it in a ZIP-file which we stream to the browser. We do code some things in the basename of the backup:

- the hostname
- the database name
- the database prefix
- the date and the time

which should be enough to distinguish nearly all backups if you happen to have a lot of different ones. Note that the URL is also encoded as a comment in the .ZIP.

The parameter `download` is currently set to 'zip'. However, we do attempt to send the plain uncompressed data if that parameter is set to 'sql' (quick and dirty). Oh well. hopefully there is enough memory to accomodate backups of moderate sized sites.

Note that we need space to compress the data; a informal test yielded that we need about 160% of the uncompressed size of the backup (tested with a small testset). Rule of the thumb for memory: the more the merrier but at least twice the size of the uncompressed backup.

output function task_logview(&\$output) *[line 291]*

Function Parameters:

- *object* **&\$output** collects output to show to user

quick and dirty logfile viewer

this constructs a table with the contents of the logtable. fields displayed are: datim, IP-address, username, logpriority and message we use a LEFT JOIN in order to get to a meaningful username rather than a numeric user_id an attempt is made to start with the last page of the logs because that would probably be the most interesting part. We paginate the log in order to keep it manageable.

- **TODO** should we allow for fancy selection mechanisms on the logfile or is that over the top?

translatetool.class.php

/program/lib/translatetool.class.php - taking care of language translations

This file defines a class for managing translations.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: translatetool.class.php,v 1.1.1.1 2011-02-01 13:00:21 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
TRANSLATETOOL_CHORE_EDIT = edit [line 34]
TRANSLATETOOL_CHORE_LANGUAGE_ADD = language_add [line 30]
TRANSLATETOOL_CHORE_LANGUAGE_EDIT = language_edit [line 32]
TRANSLATETOOL_CHORE_LANGUAGE_SAVE = language_save [line 33]
TRANSLATETOOL_CHORE_LANGUAGE_SAVE_NEW = language_savenew [line 31]
TRANSLATETOOL_CHORE_OVERVIEW = overview [line 29]
TRANSLATETOOL_CHORE_SAVE = save [line 35]
TRANSLATETOOL_PARAM_DOMAIN = domain [line 39]
TRANSLATETOOL_PARAM_LANGUAGE_KEY = language_key [line 38]
```

This parameter identifies the language.

Note: it should not be confused with the global 'language' parameter

updatelib.php

/program/lib/updatelib.php - update wizard

This file handles all system updates. The basic idea is as follows.

We assume that a previous version of Website@School is/was already correctly installed using the code in /program/[install.php](#). Using this version the school has added many, many hours of work in entering data.

Now a new version is installed, i.e. the new files (or just the updated files) are copied/uploaded to the webserver, including the file [version.php](#) and perhaps also updated versions of modules, themes, etc. This yields an error message for visitors (the infamous 'error 050') because the database version and the file version no longer match. The user logging in into admin.php is forced to attend to the job 'update', arriving here in [job_update\(\)](#).

There an overview is presented of the core version and versions of all subsystems, with the option to upgrade those that qualify. The actual work for the core version is done in [update_core\(\)](#) in this file. The actual work for modules and themes are done via the code in those subsystems. However, these are called from here.

The user is more or less _forced_ to perform the upgrade: all ways lead to job_upgrade() while the internal (database) version does not match the file version.

The upgrade should perform all necessary steps to upgrade, ending with updating the internal version number to match the file version. After that the error message '050' for visitors is gone, and admin.php no longer forces the user to come here. It is possible, however, to manually arrive here to check the updates of modules, themes, etc. but I consider that less important. That is: an upgrade of a module or theme will NOT be forced upon the user. It is wise, though to upgrade, but that is up to the user.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: updatelib.php,v 1.2 2011-02-01 14:34:34 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

TASK_UPDATE_CORE = core *[line 64]*
TASK_UPDATE_MODULE = module *[line 65]*
TASK_UPDATE_OVERVIEW = overview *[line 63]*
TASK_UPDATE_THEME = theme *[line 66]*

void function job_update(&\$output) [line 93]

Function Parameters:

- *object &\$output* collects the html output

main entry point for update wizard (called from /program/main_admin.php)

This routine takes care of executing update routines for both the core program and modeles, themes, etc. It is called automagically whenever the core program version in the database is different from the version in the file { @Ink version.php } (see also [main_admin\(\)](#)).

It can also be called manually via 'job=update'. When no specific task is specified, this routine shows the overview of versions for core, modules, themes, etc. Whenever a component is NOT up to date, an [Update] button is displayed. If a component IS up to date, we simply display the word 'OK'. This implies that when everything is up to date, the overview simply displays a list of OK's and the user is 'free to go'.

The actual updates for modules, themes, etc. is done via the various subsystems themselves, e.g. by calling `htmlpage_upgrade()` in the file `/program/modules/htmlpage/htmlpage_install.php`. The updates for the core program are actually performed from this file right here, see below for an example.

void function show_update_overview(&\$output) [line 139]

Function Parameters:

- *object &\$output* collects the html output

display an introductory text for update + status overview

void function update_core(&\$output) [line 428]

Function Parameters:

- *object &\$output* collects the html output

update the core version in the database to the version in the version.php file (the 'manifest' version)

bool function update_core_2010120800(&\$output) [line 448]

Function Parameters:

- *object* **&\$output** collects the html output

perform actual update to version 2010120800

bool function update_core_2010122100(&\$output) [*line 466*]

Function Parameters:

- *object* **&\$output** collects the html output

perform actual update to version 2010122100

bool function update_core_2011020100(&\$output) [*line 484*]

Function Parameters:

- *object* **&\$output** collects the html output

perform actual update to version 2011020100

bool function update_core_version(&\$output, \$version) [*line 261*]

Function Parameters:

- *object* **&\$output** collects the html output
- *int* **\$version** the new version number to store in config table

record the specified version number in the config table AND in \$CFG->version

This utility routine records the new version number in the config table and also adjusts the version number already in core (in \$CFG->version).

void function update_module(&\$output, \$module_id) [*line 292*]

Function Parameters:

- *object* **&\$output** collects the html output
- *int* **\$module_id** primary key for module record in modules table in database

call the module-specific upgrade routine

this routine tries to execute the correct upgrade script/function for module \$module_id. If all goes well, a success message is written to \$output (and the update is performed), otherwise an error message is written to \$output Either way the event is logged via logger().

Note that we take care not to load spurious files and execute non-existing functions. However, at some point we do have to have some trust in the file system...

array function update_status_update([\$task = NULL], [\$id = NULL]) [line 239]

Function Parameters:

- *string* **\$task** which update task do we need to do?
- *int|null* **\$id** which module/theme/etc. (NULL for core)

return an anchor tag with link to the specific update function

This utility routine returns a ready to use HTML anchor tag.

void function update_theme(&\$output, \$theme_id) [line 364]

Function Parameters:

- *object* **&\$output** collects the html output
- *int* **\$theme_id** primary key for theme record in themes table in database

call the theme-specific upgrade routine

this routine tries to execute the correct upgrade script/function for theme \$theme_id. If all goes well, a success message is written to \$output (and the update is performed), otherwise an error message is written to \$output Either way the event is logged via logger().

Note that we take care not to load spurious files and execute non-existing functions. However, at some point we do have to have some trust in the file system...

useraccount.class.php

/program/lib/useraccount.class.php - taking care of useraccounts

This file defines a class for dealing with users. Also, the global job permission constants and access control constants are defined. This file is always included, even when a visitor is anonymous (ie. not logged in).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: useraccount.class.php,v 1.1.1.1 2011-02-01 13:00:12 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
ACL_ROLE_GURU = -1 [line 99]
ACL_ROLE_INTRANET_ACCESS = 1 [line 100]
ACL_ROLE_NONE = 0 [line 98]
ACL_ROLE_PAGEMANAGER_AREAMASTER = ACL_ROLE_PAGEMANAGER_SECTIONMASTER|
PERMISSION_AREA_DROP_PAGE|
PERMISSION_AREA_ADD_PAGE|
PERMISSION_AREA_DROP_SECTION|
PERMISSION_AREA_ADD_SECTION|
PERMISSION_AREA_EDIT_AREA [line 112]
ACL_ROLE_PAGEMANAGER_CONTENTMASTER = PERMISSION_NODE_EDIT_CONTENT [line 101]
ACL_ROLE_PAGEMANAGER_PAGEMASTER = ACL_ROLE_PAGEMANAGER_CONTENTMASTER|
PERMISSION_NODE_DROP_CONTENT|
PERMISSION_NODE_ADD_CONTENT|
PERMISSION_NODE_EDIT_PAGE [line 102]
ACL_ROLE_PAGEMANAGER_SECTIONMASTER = ACL_ROLE_PAGEMANAGER_PAGEMASTER|
PERMISSION_NODE_DROP_PAGE|
PERMISSION_NODE_ADD_PAGE|
PERMISSION_NODE_DROP_SECTION|
PERMISSION_NODE_ADD_SECTION|
PERMISSION_NODE_EDIT_SECTION [line 106]
ACL_ROLE_PAGEMANAGER_SITEMASTER = ACL_ROLE_PAGEMANAGER_AREAMASTER|
PERMISSION_SITE_DROP_AREA|
PERMISSION_SITE_ADD_AREA|
PERMISSION_SITE_EDIT_SITE [line 118]
JOB_PERMISSION_ACCOUNTMANAGER = 16 [line 47]
```

This (dangerous) permission allows access to add/edit/delete users and groups (including escalate privileges)

```
JOB_PERMISSION_BACKUPTOOL = 256 [line 59]
```

This allows the user to download a backup of the database

JOB_PERMISSION_CONFIGURATIONMANAGER = 32 *[line 50]*

This permission allows the user to access the configuration manager and change the site configuration

JOB_PERMISSION_FILEMANAGER = 4 *[line 41]*

This permission allows the user to access the file manager and upload/delete files in selected places

JOB_PERMISSION_GURU = -1 *[line 32]*

Guru permissions = all permission bits are set, even the unused ones

JOB_PERMISSION_LOGVIEW = 512 *[line 62]*

This allows the user to view the contents of the log table

JOB_PERMISSION_MASK = JOB_PERMISSION_NEXT_AVAILABLE_VALUE-1 *[line 74]*

This mask can be used to isolate only the 'official' permissions from an integer value

JOB_PERMISSION_MODULEMANAGER = 8 *[line 44]*

This permission allows the user to access the module manager and configure modules

JOB_PERMISSION_NEXT_AVAILABLE_VALUE = 2048 *[line 71]*

NOTE: This quasi-permission should always be defined to be the highest permission 1

JOB_PERMISSION_PAGEMANAGER = 2 *[line 38]*

This permission allows the user to access the page manager and add/edit/delete nodes according to the user's ACLs

JOB_PERMISSION_STARTCENTER = 1 *[line 35]*

This permission is required for every user that is to logon to admin.php

JOB_PERMISSION_STATISTICS = 64 *[line 53]*

This permissions allows the user to access the site statistics

JOB_PERMISSION_TOOLS =

JOB_PERMISSION_TRANSLATETOOL|JOB_PERMISSION_BACKUPTOOL|JOB_PERMISSION_LOGVIEW|JOB_PERMISSION_UPDATE *[line 68]*

combine the permssions for the tools in a single bit mask for convenient testing

JOB_PERMISSION_TRANSLATETOOL = 128 *[line 56]*

This allows the user to translate the program, by modifying existing translations or adding new languages

JOB_PERMISSION_UPDATE = 1024 *[line 65]*

This allows the user to perform a system upgrade (see [also version check\(\)](#) and [main admin\(\)](#))

PERMISSION_AREA_ADD_PAGE = 1024 *[line 89]*

PERMISSION_AREA_ADD_SECTION = 4096 *[line 91]*

PERMISSION_AREA_DROP_PAGE = 512 *[line 88]*

PERMISSION_AREA_DROP_SECTION = 2048 *[line 90]*

PERMISSION_AREA_EDIT_AREA = 8192 *[line 92]*

PERMISSION_NODE_ADD_CONTENT = 4 *[line 79]*

PERMISSION_NODE_ADD_PAGE = 32 *[line 83]*

PERMISSION_NODE_ADD_SECTION = 128 *[line 85]*

PERMISSION_NODE_DROP_CONTENT = 2 *[line 78]*

PERMISSION_NODE_DROP_PAGE = 16 *[line 82]*

PERMISSION_NODE_DROP_SECTION = 64 *[line 84]*

PERMISSION_NODE_EDIT_CONTENT = 1 *[line 76]*

PERMISSION_NODE_EDIT_PAGE = 8 *[line 80]*

PERMISSION_NODE_EDIT_SECTION = 256 *[line 86]*

PERMISSION_SITE_ADD_AREA = 32768 *[line 95]*

PERMISSION_SITE_DROP_AREA = 16384 *[line 94]*

PERMISSION_SITE_EDIT_SITE = 65536 *[line 96]*

usermanager.class.php

/program/lib/usermanager.class.php - taking care of user management

This file defines a class for dealing with users.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: usermanager.class.php,v 1.1.1.1 2011-02-01 13:00:43 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

GROUP_SELECT_ALL_USERS = -1 [*line 30*]

this value is used to select all users rather than users from a specific group

GROUP_SELECT_NO_GROUP = 0 [*line 33*]

this value is used to select the users that are not associated with any group

waslib.php

/program/lib/waslib.php - core functions

This file provides various utility routines.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: waslib.php,v 1.1.1.1 2011-02-01 13:00:26 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

```
CAPACITY_CHAIR = 8 [line 38]
CAPACITY_CUSTOM1 = 11 [line 41]
CAPACITY_CUSTOM2 = 12 [line 42]
CAPACITY_CUSTOM3 = 13 [line 43]
CAPACITY_CUSTOM4 = 14 [line 44]
CAPACITY_CUSTOM5 = 15 [line 45]
CAPACITY_CUSTOM6 = 16 [line 46]
CAPACITY_CUSTOM7 = 17 [line 47]
CAPACITY_CUSTOM8 = 18 [line 48]
CAPACITY_CUSTOM9 = 19 [line 49]
CAPACITY_EDITOR = 9 [line 39]
CAPACITY_MEMBER = 4 [line 34]
CAPACITY_NEXT_AVAILABLE = 20 [line 50]
CAPACITY_NONE = 0 [line 30]
```

The constants **CAPACITY_*** are used for group memberships (see [accountmanagerlib.php](#)).

```
CAPACITY_PRINCIPAL = 3 [line 33]
CAPACITY_PROJECTLEAD = 5 [line 35]
CAPACITY_PUBLISHER = 10 [line 40]
CAPACITY_PUPIL = 1 [line 31]
CAPACITY_SECRETARY = 7 [line 37]
CAPACITY_TEACHER = 2 [line 32]
CAPACITY_TREASURER = 6 [line 36]
QUASI_RANDOM_DIGITS = 10 [line 253]
QUASI_RANDOM_DIGITS_UPPER = 36 [line 255]
QUASI_RANDOM_DIGITS_UPPER_LOWER = 62 [line 256]
QUASI_RANDOM_HEXDIGITS = 16 [line 254]
string function appropriate_legal_notices($high_visibility, [$m = ""]) [line 1515]
```

Function Parameters:

- *bool* **\$high_visibility** if TRUE we return a text-only link, otherwise a clickable image
- *string* **\$m** margin to improve readability of generated code

construct a link to appropriate legal notices as per AGPLv3 section 5

This routine constructs ready-to-use HTML-code for a link to the Appropriate Legal Notices, which are to be found in /program/about.html. Depending on the highvisibility flag we either generate a text-based link or a clickabel image.

The actual text / image to use depends on the global constant WAS_ORIGINAL. This constant is defined in /program/version.php and it should be TRUE for the original version of Website@School and FALSE for modified versions.

In the former case the anchor looks like 'Powered by Website@School', in the latter case it will look like 'Based on Website@School', which is in line with the requirements from the license agreement for Website@School, see /program/license.html.

IMPORTANT NOTE

Please respect the license agreement and change the definition of WAS_ORIGINAL to FALSE if you modify this program (see /program/version.php). You also should change the file '/program/about.html' and add a 'prominent notice' of your modifications.

Note: a comparable routine can be found in [install.php](#).

array function build_tree(\$area_id, [\$force = FALSE]) [*line 915*]

Function Parameters:

- *int* **\$area_id** the area to make the tree for
- *bool* **\$force** if TRUE forces reread from database (resets the cache)

construct a tree of nodes in memory

this reads the nodes in the specified area from disk and constructs a tree via linked lists (sort of). If parameter \$force is TRUE, the data is read from the database, otherwise a cached version is returned (if available).

Note that this routine also 'repairs' the tree when an orphan is detected. The orphan is automagically moved to the top of the area. Of course, it shouldn't happen, but if it does we are better off with a magically _appearing_ orphan than a _disappearing_ node.

A lot of operations in the page manager work with a tree of nodes in some way, e.g. walking the tree and displaying it or walking the tree and collecting the sections (but not the pages),

etc.

The tree starts with a 'root' with key 0 (\$tree[0]). This is the starting point of the tree. The nodes at the top level of an area are linked from this root node via the field 'first_child_id'. If there are no nodes in the area, this field 'first_child_id' is zero. The linked list is constructed by using the node_id. All nodes in an area are collected in an array. This array is used to construct the linked lists.

Every node has a parent (via 'parent_id'), where the nodes at the top level have a parent_id of zero; this points to the 'root'. The nodes within a section or at the top level are linked forward via 'next_sibling_id' and backward via 'prev_sibling_id'. A zero indicates the end of the list. Children start with 'first_child_id'. A value of zero means: no children.

The complete node record from the database is also stored in the tree. This is used extensively throughout the pagemanager; it acts as a cache for all nodes in an area.

Note that we cache the node records per area. If two areas are involved, the cache doesn't work very well anymore. However, this doesn't happen very often; only in case of moving nodes from one area to another (and even then).

- **TODO** what if we need the trees of two different areas? should the static var here be an array, keyed by area_id?
- **TODO** repairing a node doesn't really belong here, in this routine. we really should have a separate 'database repair tool' for this purpose. someday we'll fix this....

array function calculate_uri_shortcuts(\$www, \$progwww) [line 189]

Function Parameters:

- *string* **\$www** the uri (scheme / authority / path) of the directory holding config.php
- *string* **\$progwww** the uri (scheme / authority / path) corresponding with the program directory

try to eliminate the scheme and authority from the two main uri's

This tries to get rid of the scheme and the authority in 'www' and 'progwww'. If these two elements are the same, it becomes possible to use a shorter form of the uri when referencing files in 'progwww' from 'www'.

If the scheme and the authority of 'www' and 'progwww' are the same, the returned strings contain only the path elements. If scheme and authority differ, they contain the same as 'www' and 'progwww' respectively.

Examples: `www = 'http://www.example.com/site'` and `progwww = 'http://www.example.com/site/program'` yields `www_short = ''` and `wwwprog_short = '/program'`.

`www = 'http://www.example.com'` and `progwww = 'http://common.example.com/program'` yields `www_short` identical to `www` and `progwww` identical to `progwww_short`.

The purpose is to be able to generate relative links, e.g. an image in `/program/graphics/foo.jpg` can be referred to like this

```
 or  
 rather than  

```

Note that the comparison in this routine is not very fancy, it can be easily fooled to consider scheme+authority to be different. However, since this routine is only used to compare two values from `config.php`, it's not likely to cause trouble.

array function `calc_user_related_acls($user_id)` [line 1153]

Function Parameters:

- *int* **\$user_id** the user we're looking at

calculate an array with acs related to user \$user_id via group memberships

this calculates the related acs for user `$user_id`. The results are returned as an array keyed by `acl_id`. It can contain 0 or more elements. The values of the array elements are groupname/capacity-pairs. This routine is referenced from both [useraccount.class.php](#) and [usermanager.class.php](#).

string function `capacity_name($capacity)` [line 1130]

Function Parameters:

- *int* **\$capacity** numeric code of capacity

translate a numeric capacity code to a readable name

this translates a capacity code into a readable name, e.g. as an item in a dropdown list when dealing with group memberships. The actual codes are defined as constants, e.g. `CAPACITY_NONE`.

mixed function `convert_to_type($type, $value)` [line 1213]

Function Parameters:

- *string* **\$type** new type for \$value: b=bool, i=integer, s=string, etc.
- *string* **\$value** the value to convert to type \$type

convert a string to another type (bool, int, etc.)

- **TODO** perhaps change the possible values of \$type to full strings rather than 'cryptic' single letter codes. Furthermore: what do we do with invalid dates, times and date/times? For now it is a stub, returning \$value as-is. Oh well.

int function cron_send_queued_alerts([\$max_messages = 10]) [line 801]

Function Parameters:

- *int* **\$max_messages** do not send more than this number of messages

send pending messages/alerts

this goes through all the alert accounts to see if any messages need to be sent out by email. The strategy is as follows. First we collect a maximum of \$max_messages alerts in in core (1 trip to the database) Then we iterate through that collection and for every alert we

1. construct and send an email message
2. update the record (reset the message buffer and message count) (+1 trip to the database)

Locking and unlocking would be even more expensive, especially when chances of race conditions are not so big. (An earlier version of this routine went to the database once for the list of all pending alerts and subsequently twice for each alert but eventually I considered that too expensive too).

Assuming that an UPDATE is more or less atomic, we hopefully can get away with an UPDATE with a where clause looking explicitly for the previous value of the message count. If a message was added after retrieving the alerts but before updating, the message count would be incremented (by the other process) which would prevent us from updating. The alert would be left unchanged but including the added message. Worst case: the receiver gets the same list of alerts again and again. I consider that a fair trade off, given the low probability of it happening. (Mmmm, famous last words...)

Bottom line, we don't do locking in this routine.

Note that we add a small reminder to the message buffer about us processing the alert and sending a message. However, we don't set the number of messages to 1 because otherwise that would be the signal to send this message the next time. We don't want to send a message every `$cron_interval` minutes basically saying that we didn't do anything since the previous run. (Or is this a feature after all?)

Failures are logged, success are logged as `LOG_DEBUG`.

array/bool function `get_area_records([$forced = FALSE])` [line 1108]

Function Parameters:

- *bool* **\$forced** if TRUE forces reread from database (resets the cache)

retrieve a list of all available area records keyed by area_id

this returns a list of area-records or FALSE if no areas are available. The list is cached via a static variable so we don't have to go to the database more than once for this. Note that the returned array is keyed with `area_id` and is sorted by `sort_order`. Also note that this list may include areas for which the current user has no permissions whatsoever.

mixed function `get_parameter_int($name, [$default_value = NULL])` [line 482]

Function Parameters:

- *string* **\$name** the name of the parameter to retrieve the value of
- *mixed* **\$default_value** the value to return if parameter was not specified

return an integer value specified in the page request or default value if none

mixed function `get_parameter_string($name, [$default_value = NULL])` [line 497]

Function Parameters:

- *string* **\$name** the name of the parameter to retrieve the value of
- *mixed* **\$default_value** the value to return if parameter was not specified

return an (unquoted) string value specified in the page request or default value if none

bool|array function get_properties([\$tablename = 'config'], [\$where = '']) [*line 116*]

Function Parameters:

- *string* **\$tablename** the name of the table holding the properties
- *array|string* **\$where** which records do we need to select

retrieve typed properties (name-value-pairs) from a table

this retrieves the fields 'name', 'value' and 'type' from all records from \$tablename that satisfy the condition in \$where. The values, which are stored as strings in the database, are converted to their proper value type and stored in the resulting array, keyed by name. The following types are recognised:

- b = boolean
- d = date ('yyyy-mm-dd', handled like a string)
- dt = datetime ('yyyy-mm-dd hh:mm:ss', handled like a string)
- f = float
- i = integer
- s = string
- t = time ('hh:mm:ss', handled like a string)

Note that we currently do not validate these properties, the assumption is that the values are valid (or empty).

int|null function get_requested_area() [*line 433*]

get the number of the area the user requested or null if not specified

See discussion of [get_requested_node\(\)](#). We use separate routine because we may want to support `index.php/aaa/nnn/....` instead of `index.php?area=aaa&node=nnn&...`

string|null function get_requested_filename() [*line 465*]

get the name of the requested file

See discussion of [get_requested_node\(\)](#). Files are served via `/file.php` via a comparable mechanism: either

`http://localhost/file.php/path/to/filename.ext`

OR

`http://localhost/file.php?file=/path/to/filename.ext`

This routine extracts the `'/path/to/filename.ext'` part.

int|null function `get_requested_node()` [*line 408*]

get the number of the node the user requested or NULL if not specified

This routine exists because nodes and areas are so central to the whole idea of WAS.

Purpose is to retrieve any requested `node_id` from the parameters submitted by the user. As a rule this works via name-value-pairs, something like this: `index.php?area=aaa&node=nnn`. However, if the webserver is configured correctly, we can also accept `index.php/aaa/nnn/...` or `index.php/nnn/...` which is more proxy-friendly. Using a generic routine like `get_parameter_int()` would not be sufficient in that case, so there.

Note that the same proxy-friendly 'trick' is used to determine the filename of a file that needs to be served via [file.php](#) (see [get_requested_filename\(\)](#)).

Note that we first look at the proxy-friendly variant. If that doesn't work, we resort to the conventional way of `index.php?node=nnn`. Also note that the order of the `path_info` is important. If there is just a single numeric path component, we assume that it is the node value; if there are two numerics our assumption is that the first one is the area id and the second one the node id.

Note that this routine does not validate the requested node in any way other than making sure that IF it is specified, it is an integer value. For all we know it might even be a negative value.

int function `get_unique_number([$\$increment = TRUE$])` [*line 1476*]

Function Parameters:

- *bool* **$\$increment$** optional indicates whether the static counter must be incremented

a small utility routine that returns a unique integer

this generates a unique number (starting at 1). This number is guaranteed to be unique during this http-request (or at least until the static variable `$\$id$` overflows, but that takes a while). If the optional parameter `$\$increment$` is `FALSE`, the latest id returned is returned again.

array function `get_user_groups($\$user_id$)` [*line 1182*]

Function Parameters:

- *int* **$\$user_id$** the user we're looking at

retrieve the records of the groups of which user $\$user_id$ is a member

- **Uses \$DB**

int function ini_get_int(\$variable) [*line 1311*]

Function Parameters:

- *string* **\$variable** name of the variable to retrieve, e.g. 'upload_max_filesize'

return an integer (bytecount) value from PHP ini

bool function is_expired(\$node_id, &\$tree) [*line 1070*]

Function Parameters:

- *int* **\$node_id**
- *array* **&\$tree** family tree

determine if any of the ancestors or \$node_id itself is already expired

This climbs the tree upward, starting at \$node_id, to see if any nodes are expired. If an expired node is detected, TRUE is returned. If none of the nodes are expired, then FALSE is returned.

Note that this routine looks strictly at the expiry property, it is very well possible that a node is under embargo, see [is_under_embargo\(\)](#).

Also note that this routine currently also tries to 'fix' the node database when a circular reference is detected. This doesn't really belong here, but for the time being it is convenient to have this auto-repair mechanism here. The node that is fixed is the section we are looking at after MAXIMUM_ITERATIONS tries, which is not necessarily the node we started with.

- **TODO** this function also 'repairs' circular references. This should move to a separate tree-repair function but for the time being it is "convenient" to have automatic repairs...
- **Uses \$DB**

bool function `is_under_embargo(&$tree, $node_id)` [line 1022]

Function Parameters:

- *array* **&\$tree** family tree
- *int* **\$node_id**

determine if any of the ancestors or \$node_id itself is under embargo

This climbs the tree upward, starting at \$node_id, to see if any nodes are under embargo. If an embargo'ed node is detected, TRUE is returned. If none of the nodes are under embargo, then FALSE is returned.

Note that this routine looks strictly at the embargo property, it is very well possible that a node is expired, see [is_expired\(\)](#).

Also note that this routine currently also tries to 'fix' the node database when a circular reference is detected. This doesn't really belong here, but for the time being it is convenient to have this auto-repair mechanism here. The node that is fixed is the section we are looking at after MAXIMUM_ITERATIONS tries, which is not necessarily the node we started with.

- **TODO** this function also 'repairs' circular references. This should move to a separate tree-repair function but for the time being it is "convenient" to have automatic repairs...
- **Uses** \$DB

string function `javascript_alert($message)` [line 292]

Function Parameters:

- *string* **\$message** message to display

message a message and generate a javascript alert()

- **Usedby** [login_page_close\(\)](#)

bool function lock_record(\$id, &\$lockinfo, \$tablename, \$pkey, \$locked_by, \$locked_since) [*line 646*]

Function Parameters:

- *int* **\$id** the primary key of the record to lock
- *array* **&\$lockinfo** returns information about the session that already locked this record
- *string* **\$tablename** the name of the table
- *string* **\$pkey** name of the field holding the serial (pkey)
- *string* **\$locked_by** name of the field to hold our session_id indicating we locked the record
- *string* **\$locked_since** name of the field holding the datetime when the lock was obtained

put a (co-operative) lock on a record

this tries to set the co-operative) lock on the record with serial (pkey) \$id in table \$tablename by setting the \$locked_by field to our own session_id. This is the companion routine of [lock_release\(\)](#).

The mechanism of co-operative locking works as follows. Some tables (such as the 'nodes' table) have an int field, e.g. 'locked_by_session_id'. This field can either be NULL (indicating that the record is not locked) or hold the primary key of a session (indicating that the record is locked and also by which session).

Obtaining a lock boils down to updating the table and setting that field to the session_id. As long as the underlying database system guarantees that execution of an UPDATE statement is not interrupted, we can use UPDATE as a 'Test-And-Set'-function. According to the docentation MySQL does this.

The procedure is as follows.

1. we try to set the locked_by-field to our session_id on the condition that the previous value of that field is NULL. If this succeeds, we have effectively locked the record.
2. If this fails, we retrieve the current value of the field to see which session has locked it. If this happens to be us, we had already locked the record before and we're done.
3. If another session_id holds the lock, we check for that session's existence. If it still exists, we're out of luck: we can't obtain the lock.
4. If that other session does no longer exist, we try to replace that other session's session_id with our own session_id, once again using a single UPDATE (avoiding another race condition). If that succeeds we're done and we have the lock; if it failes we're also done but without lock.

If locking the record fails because the record is already locked by another session, this routine returns information about that other session in `$lockinfo`. It is up to the caller to use this information or not.

Note. A record can stay locked if the webbrowser of the locking session has crashed. Eventually this will be resolved if the crashed session is removed from the sessions table. However, the user may have restarted her browser while the record was locked. From the new session it appears that the record is still locked. This may take a while. Mmmmm... The other option is to lock on a per-user basis rather than per-session basis. Mmmm... Should we ask the user to override the session if it happens to be the same user? Mmm. put it on the todo list. (A small improvement might be to call the garbage collection between step 2 and 3. Oh well).

- **TODO** we need to resolve the problem of crashing browsers and locked records
- **TODO** perhaps we can save 1 trip to the database by checking for something like `UPDATE SET locked_by = $session_id WHERE (id = $id) AND ((locked_by IS NULL) OR (locked_by = $session_id))` but I don't know how many affected rows that would yield if we already had the lock and effectively nothing changes in the record. (Perhaps always update atime to force 1 affected row?)
- **TODO** do we need a 'force lock' option to forcefully take over spurious locks?
- **Usedby** [lock_release_node\(\)](#)
- **Usedby** [lock_record_node\(\)](#)

bool function lock_record_node(\$node_id, &\$lockinfo) [line 566]

Function Parameters:

- *int* **\$node_id** the primary key of the node to lock
- *array* **&\$lockinfo** returns information about the session that already locked this record

get record lock on a node

this is a wrapper around [lock_record\(\)](#) for locking nodes.

- Uses [lock_record\(\)](#)

bool function lock_release(\$id, \$tablename, \$pkey, \$locked_by, \$locked_since) [line 735]

Function Parameters:

- *int* **\$id** the primary key of the record to unlock
- *string* **\$tablename** the name of the table
- *string* **\$pkey** name of the field holding the serial (pkey)
- *string* **\$locked_by** name of the field holding the session_id of the session that locked the record
- *string* **\$locked_since** name of the field holding the datetime when the lock was obtained

unlock a record that was previously successfully locked

this removes the co-operative) lock on the record with serial (pkey) \$id in table \$tablename by setting the \$locked_by field to NULL. This is the companion routine of [lock_record\(\)](#).

bool function lock_release_node(\$node_id) [line 579]

Function Parameters:

- *int* **\$node_id** the primary key of the node record to unlock

release lock on a node

this is a wrapper around [lock_release\(\)](#) for unlocking nodes.

- Uses [lock_record\(\)](#)

bool function logger(\$message, [\$priority = LOG_INFO], [\$user_id = ""]) [line 354]

Function Parameters:

- *string* **\$message** the message to write to the log

- *int* **\$priority** loglevel, see PHP-function syslog() for a list of predefined constants
- **\$user_id**

a simple function to log information to the database 'for future reference'

This adds a message to the table log_messages, including a time, the remote address and (of course) a message. See also the standard PHP-function syslog(). We use the existing symbolic constants for priority. Default value is LOG_INFO.

Note that messages with a priority LOG_DEBUG are only written to the log if the global parameter \$CFG->debug is TRUE. All other messages are simply logged, no further questions asked.

If the caller does not provide a user_id, this routine attempts to read the user_id from the global \$_SESSION array, i.e. we try to link events to a particular user if possible.

- **TODO** should we make this configurable and maybe log directly to syslog (with automatic logrotate) or do we want to keep this 'self-contained' (the webmaster can read the table, but not the machine's syslog)?
- **Usedby** [login_send_bypass\(\)](#)
- **Usedby** [login_send_laissez_passer\(\)](#)
- **Uses** \$CFG

string function magic_unquote(\$value) [*line 83*]

Function Parameters:

- *string* **\$value** a string value that is conditionally unescaped

this circumvents the 'magic' in magic_quotes_gpc() by conditionally stripping slashes

Magic quotes are a royal pain for portability. If magic quotes are enabled, this function reverses the effect. There are three PHP-parameters in php.ini affecting the magic:

- the directive 'magic_quotes_runtime'
- the directive 'magic_quotes_gpc'
- the directive 'magic_quotes_sybase'

This routine deals with undoing the effect of the latter two. The effect of magic_quotes_runtime can be undone via set_magic_quotes_runtime(0). This is done once

at program start (See [initialise\(\)](#) in [init.php](#)).

This routine should be used to unquote strings from `$_GET[]`, `$_POST[]` and `$_COOKIE` whenever they are needed.

Important note: because third party subsystems may deal with magic quotes on their own, it is a Bad Idea[tm] to globally replace the contents of `$_GET[]`, `$_POST[]` and `$_COOKIE` with the unescaped values once at program start. Any subsystem would be confused if `magic_quotes_gpc()` indicates that the magic is in effect whereas in reality the magic was already undone at program start. Yes, this yields a performance penalty, but this magic was a mess right from the start. Hopefully PHP6 will get rid of this magic for once and for all...

int function performance_get_queries() [*line 538*]

return the number of database queries that was executed

- **Uses \$DB**

double function performance_get_seconds() [*line 550*]

return the script execution time

- **TODO** maybe we should get rid of this \$PERFORMANCE object, because it doesn't do that much anyway

void function quasi_random_string(\$length, [\$candidates = 36]) [*line 276*]

Function Parameters:

- *int* **\$length** length of the string to generate
- *int* **\$candidates** number of candidate-characters to choose from

generate a string with quasi-random characters

This generates a string of `$length` quasi-random characters. The optional parameter `$candidates` determines which characters are eligible. Popular choices for `$candidates` are:

- 10 (minimum): use only digits from 0,...,9
 - 16: use digits 0,...,9 or letters A,...,F
 - 36 (default): use digits 0,...,9 or letters A,...,Z
 - 62: use digits 0,...,9 or letters A,...,Z or letters a,...,z
- If \$candidates is smaller than 10, 10 is used, if \$candidates is greater than 62 62 is used.

string function quoted_printable(\$s, [\$textmode = TRUE], [\$newline = "\n"], [\$max_length = 76]) [*line* 1415]

Function Parameters:

- *string* **\$s** source string
- *bool* **\$textmode** TRUE means newlines count as hard line breaks, FALSE is binary data
- *string* **\$newline** native character indicating end of line
- *int* **\$max_length** indicates the limit for output lines (excluding the CRLF)

convert string \$s from native format to quoted printable (RFC2045)

this converts the input string \$s to quoted printable form as defined in RFC2045 (see <http://www.ietf.org/rfc/rfc2045.txt>). By default this routine assumes a line-oriented text input. This can be overruled by calling the routine with the parameter \$textmode set to FALSE: in that case the input is considered to be a binary string with no embedded newlines.

The routine assumes that the input lines are delimited with \$newline. By default this parameter is a LF (Linefeed) but it could be changed to another delimiter using the function parameter \$newline.

According to RFC2045 the resulting output lines should be no longer than 76 bytes, even though it is very well possible to use shorter lines. This can be done by setting the parameter \$max_length to the desired value. Note that this value is forced to be in the range 4,...,76.

The encoding is defined in section 6.7 of RFC2045 with these five rules.

(1) General 8bit representation: any character may be represented as "=" followed by two uppercase hexadecimal digits.

(2) Literal representation characters "!" to "~" but excluding the "=" may represent themselves.

(3) White space Space " " and tab "\t" at the end of a line must use rule (1); in all other cases either rule (1) or (2) may be applied.

(4) Line breaks The (hard) line breaks in the input must be represented using "\r\n" in the output.

(5) Soft line breaks Output lines may not be longer than 76 bytes. This can be enforced by inserting a soft line break (the string `"=\r\n"`) in the output. This soft line break will disappear once the encoded string is decoded.

The basic conversion algorithm is constructed using two important variables:

- an integer value (`$remaining`) indicating the number of bytes left in the current output line
- a boolean flag (`$next_is_newline`) indicating if the next input character is a `$newline`

The variable `$remaining` keeps track of situations where the current character (either as (1) General 8bit representation or (2) Literal representation) might not fit on the current line (eg. 2 bytes left requires an 8bit representation to be moved to the next output line). The flag `$next_is_newline` is used to make the best possible use of the available remaining space in the output, eg. if the current character is exactly as long as the remaining space, we can output that character on the current output line, because we are sure that it is the last character on the current output line so there cannot be a soft return next.

Note that spaces (ASCII 32) and tabs (ASCII 9) are treated differently depending on their position in the line. The rule is that both should be represented as `"=20"` or `"=09"` at the end of an input line and that it is allowed to use `" "` or `"\t"` when NOT at the end of an input line. In the latter case, the output line will always end with a soft line break `"=\r\n"` which makes sure that there are not trailing spaces/tabs in the output line anyway.

Also note that the end of the input `$s` is also flagged via setting `$next_is_newline`. This is an optimisation which treats spaces and tabs at the end of the input as if they were at the end of an input line, ie. converting to `"=20"` or `"=09"`. This means that the output will never end with a space or a tab, even if the input does.

Note that in case of a binary conversion the input character(s) that might otherwise indicate a newline are to be considered as binary data. However, if the data is completely binary, it probably doesn't make sense to use Quoted-Printable in the first place (base64 would probably be a better choice).

Reference: see <http://www.ietf.org/rfc/rfc2045.txt>.

- **TODO** should we change the code to accomodate the canonical newline CRLF in the input?

nothing function `redirect_and_exit($url, [$message = "])` [line 511]

Function Parameters:

- *string* **\$url** the url to redirect to
- **\$message**

redirect to another url by sending an http header

string function `replace_crlf($multiline_string, [$replacement = ""])` [line 309]

Function Parameters:

- *string* **\$multiline_string** the multiline string to strip
- *string* **\$replacement** (optional) the string to replace newlines

unfold a possible multiline string

This removes all linefeeds and carriage returns from a string Typical use would be to strip a subject line in a mailmessage from newlines which might interfere with proper sending of mail headers.

string function `sanitise_filename($filename)` [line 1280]

Function Parameters:

- *string* **\$filename** the string to sanitise

sanitise a string to make it acceptable as a filename/directoryname

this routine analyses and maybe converts the input string as follows:

- all leading and trailing dots, spaces, dashes, underscores, backslashes and slashes are removed
- all embedded spaces, backslashes and slashes are converted to underscores
- only letters, digits, dots, dashes or underscores are retained
- all sequences of 2 or more underscores are replaced with a single underscore
- finally all 'forbidden' words (including empty string) get an underscore prefixed

Note that this sanitising only satisfies the basic rules for filenames; creating a new file with a sanitised name may still clash with an existing file or subdirectory.

Also note that a full pathname will yield something that looks like a simple filename without directories or drive letter: C:\Program Files\Apache Group\htpasswd becomes C_Program_Files_Apache_Group_htpasswd and /etc/passwd becomes etc_passwd. Also this routine makes a URL look like a filename: http://www.example.com becomes

http_www.example.com.

Finally note that we don't even attempt to transliterate utf8-characters or any other characters between 128 and 255; these are simply removed.

- **TODO** should we check for overlong UTF-8 encodings: C0 AF C0 AE C0 AE C0 AF equates to `./.` or is that dealt with already by filtering on letters/digits and embedded dots/dashes/underscores?

bool/long function string2time(\$timestring) [line 225]

Function Parameters:

- *string* **\$timestring** date/time in the form yyyy-mm-dd hh:mm:ss

convert a string representation of a date/time to a timestamp

this is a crude date/time parser. We collect digits and convert to integers. With the integers we fill an array with at least 6 integers, corresponding to year, month, day, hours, minutes and seconds. If there are less than six numbers in the source string the value 0 is used. for the remaining elements. Note that a number in this context is always a non-negative number because a dash (or minus) is considered a delimiter.

Note that valid date/time values are limited to how many seconds can be represented in a signed long integer, where 0 equates to 1970-01-01 00:00:00 (the Unix epoch). The upper limit for a 32-bit int is some date in 2038 (only 30 years from now).

string function t(\$phrase_key, [\$full_domain = "], [\$replace = "], [\$location_hint = "], [\$language = "]) [line 326]

Function Parameters:

- *string* **\$phrase_key** indicates the phrase that needs to be translated
- *string* **\$full_domain** (optional) indicates the text domain (perhaps with a prefix)
- *array* **\$replace** (optional) an assoc array with key-value-pairs to insert into the translation
- *string* **\$location_hint** (optional) hints at a directory location of language files
- *string* **\$language** (optional) target language

translation of phrases via a function with a very short name

This is only a wrapper function for `$LANGUAGE->get_phrase()`

- **Uses** `$LANGUAGE`

zip.class.php

/program/lib/zip.class.php - create simple ZIP-archives

This file implements class Zip which allows for creating ZIP-archives on the fly

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: zip.class.php,v 1.1.1.1 2011-02-01 13:00:14 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

ZIP_TYPE_BUFFER = buffer *[line 32]*
ZIP_TYPE_FILE = file *[line 30]*
ZIP_TYPE_NONE = ' *[line 29]*
ZIP_TYPE_STREAM = stream *[line 31]*

main_admin.php

/program/main_admin.php - workhorse for site maintenance

This file deals with the administrator interface program for site maintenance. It is included and called from /admin.php.

The work is done in [main_admin\(\)](#).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: main_admin.php,v 1.1.1.1 2011-02-01 13:00:08 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

JOB_ACCOUNTMANAGER = accountmanager *[line 55]*

This is used to dispatch the account manager (users and groups)

JOB_CONFIGURATIONMANAGER = configurationmanager *[line 58]*

This is used to dispatch the configuration manager

JOB_FILEBROWSER = filebrowser *[line 43]*

This is used to dispatch the file manager in file browser mode (used with FCK Editor)

JOB_FILEMANAGER = filemanager *[line 40]*

This is used to dispatch the file manager

JOB_FLASHBROWSER = flashbrowser *[line 49]*

This is used to dispatch the file manager in flash browser mode (used with FCK Editor)

JOB_IMAGEBROWSER = imagebrowser *[line 46]*

This is used to dispatch the file manager in image browser mode (used with FCK Editor)

JOB_MODULEMANAGER = modulemanager *[line 52]*

This is used to dispatch the module manager

JOB_PAGEMANAGER = pagemanager *[line 37]*

This is used to dispatch the page manager

JOB_STARTCENTER = start *[line 34]*

This is used to dispatch the startcenter job

JOB_STATISTICS = statistics *[line 61]*

This is used to dispatch the statistics

JOB_TOOLS = tools *[line 64]*

This is used to dispatch the tool manager

JOB_UPDATE = update *[line 67]*

This is used to dispatch the update manager

void function add_javascript_popup_function(&\$output, [\$m = ""]) *[line 461]*

Function Parameters:

- **&\$output**
- **\$m**

add javascript code that implements a popup to the header part of the page

void function add_javascript_select_url_function(&\$output, [\$m = ""]) *[line 493]*

Function Parameters:

- **&\$output**
- **\$m**

add javascript code that implements a url selection (used in integration with FCKeditor)

void/int function admin_continue_session() *[line 421]*

continue the session from the previous call OR exit

This tries to resume the session that was initiated before (when the user logged in

successfully). If the session cannot be resumed, we logout the user, show the login screen and exit. In other words: this routine guarantees that a valid session exists if and when this routine returns. If the routine returns, it returns the user_id.

- Uses [dbsessionlib.php](#)
- Uses \$CFG;

void/int function admin_login(\$step) [*line 391*]

Function Parameters:

- *mixed* \$step the step in the login procedure

perform a step in the login procedure

This routine may not return at all. If it returns, the user is logged in successfully and the return value is the user_id (unique identification for the user, pkey in users table). The steps in the login procedure are defined in [loginlib.php](#).

- Uses [loginlib.php](#)
- Uses \$CFG

void function admin_logout_and_exit() [*line 369*]

logout the user and exit

This logs out the user (ie kills the session) If there is an error (ie, there was no session in the first place) indicated by was_logout() returning, we unconditionally show a login dialog and exit. So, this routine never returns.

- Uses [loginlib.php](#)

- **Uses** \$CFG

void function admin_show_login_and_exit([\$message = ""]) *[line 450]*

Function Parameters:

- **\$message**

show login dialog and exit

- **Uses** [loginlib.php](#)
- **Uses** \$CFG

string function get_versioncheck_url() *[line 518]*

construct URL for version check against the project's website

this constructs the URL for checking the installed version against the current version on the project's website. The remote site will respond with a readable text and the user can decide to act on the information (or not). We don't want to force any upgrades etc.

void function job_start(&\$output) *[line 306]*

Function Parameters:

- *object* **&\$output** output collector

generate the start centre page

This is the handler for the start centre. This is the only handler that is NOT included from another file. Basically it shows a screen with hints on how to proceed with this program.

- **TODO** this routine is a stub

void function main_admin() [line 105]

main program for site maintenance

This is the main administrator program. First step is to deal with users logging in or out. If a user is not logged in, a login dialog is displayed. If a user is logged in but has no admin privileges, she is redirected to the public site (ie. index.php).

Once we have established that the user is an administrator, we setup an output collecting object and see what the user wants us to do by interpreting the parameter 'job'. If the user has access to the specified job, the corresponding code is included and the main routine of that handler is called. It is then the responsibility of that handler to further decide what needs to be done. After the handler returns, the collected output is sent to the user. This includes the main navigation (i.e. links to the various 'managers') and also the menu and the content generated by the handler.

If the user has no privilege to access a particular manager, an error message is displayed in both the message area and the content area. This makes it clear to the user that access is denied. Note that the inaccessible items are displayed in the main navigation via 'dimmed' (light-grey) links or black/white images. By showing these 'dimmed' links, the user will be aware that there is more than just what she is allowed to see. This is more transparent than suppressing items and keeping them secret.

- **TODO** should we cater for a special 'print' button + support for a special style sheet for media="print"?
- **Uses** \$USER;
- **Uses** \$LANGUAGE;
- **Uses** \$CFG;

main_cron.php

/program/main_cron.php - take care of recurring jobs

This file deals with executing cron jobs. It is included and called from /cron.php.

The work is done in [main_cron\(\)](#).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: main_cron.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

void function main_cron() [*line 32*]

main_file.php

/program/main_file.php - workhorse for serving files

This file deals with serving files It is included and called from /file.php.

The work is done in [main_file\(\)](#).

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: main_file.php,v 1.1.1.1 2011-02-01 13:00:06 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

void/bool function download_source(\$component) [*line 396*]

Function Parameters:

- *string* **\$component** either 'program' or 'manual' or 'languages'

construct a zipfile with the current source and stream it to the visitor

this routine streams a ZIP-archive to the visitor with either the current websiteatschool program code or the selected manual. This routine is necessary to comply with the provisions of the program license which basically says that the source code of the running program must be made available.

Note that it is not strictly necessary to also provide the manual, but this routine can do that nevertheless.

Note that we take special care not to download the (private) data directory \$CFG->datadir. Of course the datadirectory should live outside the document root and certainly outside the /program directory tree, but accidents will happen and we don't want to create a gaping security hole.

If there are severe errors (e.g. no manual is available for download or an invalid component was specified) the program exist immediately with a 404 not found error. Otherwise the ZIP-archive is streamed to the user. If all goes well, we return TRUE, if there were errors we immediately return TRUE (without finishing the task at hand other than a perhasp futile attempt to properly close the ZIP-archive). The last error message from the Zip is logged.

- Uses [download_source_tree\(\)](#)

bool function download_source_tree(&\$zip, \$path, \$vpath, &\$excludes) [line 528]

Function Parameters:

- *object* **&\$zip** Zip-archive
- *string* **\$path** physical directory to add to archive
- *string* **\$vpath** virtual pathname for this physical directory
- *array* **&\$excludes** array with 'forbidden' subdirectories

workhorse function to recursively add most of a tree to a ZIP-archive

this routine recursively adds the tree starting at \$path to the opened archive \$zip. If a directory is in the list of excluded directories in \$excludes it is skipped.

- Usedby [download_source_tree\(\)](#)
- Usedby [download_source\(\)](#)
- Uses [download_source_tree\(\)](#)

void function error_exit404([\$filename = ""]) [line 363]

Function Parameters:

- *string* **\$filename** the file we were looking for and could not find

exit with a 404 not found error

void function main_file() [line 94]

main program for serving files

this routine is called from /file.php.

This routine is responsible for serving files to the visitor. These files are stored in a (virtual) file hierarchy that looks like this.

```
/areas/areaname
  /another
  /stillmore
...
/users/username
  /another
  /stillmore
...
/groups/groupname
  /another
  /stillmore
...
/websiteatschool/program
  /manual
  /languages
```

This structure maps to the real file system as follows. The (virtual) directories /areas, /users and /groups correspond to the physical directories {`$CFG->datadir`}/areas, {`$CFG->datadir`}/users and {`$CFG->datadir`}/groups respectively. The subdirectories correspond to a (unique) area, user or group and serve as a file repository for that area, user or group.

The (virtual) top-level directory /websiteatschool is a special case. It is used to serve the currently running website program code and the user-defined translations of active languages.

Before any file is transmitted to the visitor the access privileges are checked. The following rules apply.

Access control for the /areas subdirectory

- an area must be active before any files are served
 - the visitor must have access to the private area if files are to be served
 - non-existing files yield a 404 Not Found error
 - non-existing areas also yield a 404 Not Found error
 - if the visitor has no access to the private area, also a 404 Not Found error is returned
- Access control for /users and /groups

- a user/group must be active before any files are served
 - non-existing users/groups yield 404 Not Found
 - non-existing files in existing directories also yield 404 Not Found
- Access control for /websiteatschool

- there is no limit on downloading the currently active program code or user-defined

- **TODO** the check on '/../' is inconclusive if the \$path is encoded in UTF-8: the overlong sequence 2F C0 AE 2E 2F eventually yields 2F 2E 2E 2F or '/../'. Reference: RFC3629 section 10.

bool/int function `readfile_chunked($path)` [line 568]

Function Parameters:

- *string* **\$path** fully qualified path of the file to send

send a file to the visitor's browser in chunks

This sends the file \$path to the browser in manageable chunks.

string function `rfc1123date([$t = 0])` [line 263]

Function Parameters:

- *int* **\$t** the date/time value to use, or 0 for current time

generate an RFC1123-compliant date/time stamp

This constructs a date/time stamp that is a fixed-length subset of RFC1123. This is the preferred format in HTTP (see RFC2616 section 3.3).

The format is as follows: `rfc1123-date = wkday " , " SP date SP time SP "GMT"`

`date` = 2DIGIT SP month SP 4DIGIT ; day month year (e.g., 02 Jun 1982)

`time` = 2DIGIT ":" 2DIGIT ":" 2DIGIT ; 00:00:00 - 23:59:59

`wkday` = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"

`month` = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" |
"Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"

If \$timevalue is less or equal to zero, the current time is used, otherwies \$timevalue is interpreted as a standard unix timestamp.

void function `send_file_from_datadir($file, $name, [$mimetype = "], [$ttl = 86400], [$download = FALSE])` [line 309]

Function Parameters:

- *string* **\$file** name of the file to send relative to \$CFG->datadir
- *string* **\$name** filename to suggest to the visitor/visitor's browser
- *string* **\$mimetype** the mime type of the file; if not specified we use an educated guess
- *int* **\$ttl** time to live (aka maximum age) in seconds, 0 implies file is not cacheable
- *bool* **\$download** if TRUE we try to force a download

the designated file is sent to the visitor

This transmits the file {\$CFG->datadir}\$file from the data directory to the visitor's browser, suggesting the name \$name. The file is transmitted in chunks (see [readfile_chunked\(\)](#)).

Several different variations are possible.

- by specifying a Time To Live of 0 seconds, this routine tries hard to defeat any caching by proxies
- if the download flag is TRUE, this routine tries to prevent the visitor's browser to render the file in-line suggesting downloading instead
Quirks
- There appears to be a problem with Internet Explorer and https:// and caching which requires a specific workaround. We simply check for 'https:' or 'http'.
- Adobe Acrobat Reader has a bad track record of infecting user's computers with malware when PDF's are rendered in-line. Therefore we force download for that kind of files.
- It is not easy to determine the exact mime type of files without resorting to a complex shadow-filesystem or a metadata table in the database. Therefore we 'guess' the mime type, either based on the information provided by the fileinfo PHP-module, or simply based on the extension of \$file (which is not very reliable, but we have to do something). See [get_mimetype\(\)](#) for details.

- Uses [get_mimetype\(\)](#)

main_index.php

/program/main_index.php - workhorse for visitor interface

This file deals with the visitor interface. It is included and called from /index.php.

The work is done in [main_index\(\)](#).

Parameters are passed like index.php?parm=val. Here is an overview of recognised global parameters that are handled here (and not in a module).

- **logout** - used to end a user's session (no need for a value, just the param is enough)
- **login=i** - step i of the login procedure
- **area=a** - indicates which area to access; if specified it should match the node, if that is specified
- **node=n** - indicates which node to access; if specified it should match the area, if that is specified
- **language=xx** - indicates the language to use; xx is a valid language code like 'en', 'de', 'fr' or 'nl'

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: main_index.php,v 1.1.1.1 2011-02-01 13:00:09 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** add the performance results in a HTML-comment if not CFG->debug, in sight otherwise
- **License** [GNU AGPLv3+Additional Terms](#)

bool|array function calculate_area(\$requested_area, \$requested_node) [*line 233*]

Function Parameters:

- *int|null* **\$requested_area** the area the user specified or NULL if no area specifically requested
- *int|null* **\$requested_node** the node the user specified or NULL if no node specifically requested

try to retrieve a valid area record based on values of requested area and requested node

this determines which area to use. If the user specifies nothing (no area, no node), we

simply go for the default area or the first available area. If the user does specify an area and/or a node, we use that information to get to the area. Note that if the user specifies both area and node, the two should match. That is: you cannot specify a node from area X and also request area Y: that yields no results. If only a node is specified, the area is calculated from the area to which the node belongs.

We let the database do most of the work by constructing and executing an appropriate SQL-statement.

- **Uses \$DB**

bool/int function calculate_default_page(&\$tree, \$subtree_id) [line 326]

Function Parameters:

- *array* **&\$tree** a reference to the complete tree in the area of interest
- *int* **\$subtree_id** the place where we need to start looking (usually the first_child_id of the parent)

try to find a default page within a subtree of pages and sections

this walks the tree \$tree starting at \$subtree_id looking for a default page. We give it three tries. First we look for a default node in the section of which \$subtree_id is the first node. If we find a page, we're done, if we find a section we descend into that subtree. If there still is no default page, we go look for any page in the initial set of nodes. If that too doesn't yield a page, we descend into the subtrees. If THAT doesn't yield a page we give up and return FALSE, indicating no page to be found.

bool/int function calculate_node_id(&\$tree, \$area_id, \$requested_node) [line 293]

Function Parameters:

- *array* **&\$tree** a reference to the complete tree in area \$area_id
- *int* **\$area_id** the area where we are looking for a node
- *int/null* **\$requested_node** the node_id the user requested or NULL if none was specified

calculate and validate the node_id to display

this tries to determine a valid node to display based on the node the user requested and

the area that the user may or may not have requested.

Basic assumption is that the visitor has indeed view access to area \$area_id. This means that the user is allowed to see the nodes in this area that are not under embargo (and not expired). We do have a complete overview of all nodes in this area in the array \$tree. (See [build_tree\(\)](#) for more information about the tree structure)

The parameter \$requested_node is either an integer, indicating the user explicitly specified a node number in the page request, or null, indicating that the user did not explicitly specify a node. In the latter case the user may or may not have explicitly requested an area.

There are several cases we need to handle - if no node is explicitly requested, we need to identify the default page in the area - if the node is under embargo the node does not exist (from the POV of the user) - if the requested node is a section, we need to identify the default page in that section

void function main_index() [*line 50*]

main program for visitors

this routine is called from /index.php. It is the main program for visitors.

- **TODO** cleanup login/logout-code

bool|array function module_load_view(\$module_id) [*line 428*]

Function Parameters:

- *int* \$module_id indicates which module to load

load the visitor/view interface of a module in core

this includes the 'view'-part of a module via 'require_once()'. This routine first figures out if the view-script file actually exists before the file is included. Also, we look at a very specific location, namely: /program/modules/<modulename>/<module_view_script> where <modulename> is retrieved from the modules table in the database.

Note that if modulename would somehow be something like `"../..../etc/passwd\x00"`, we could be in trouble...

- **TODO** should we sanitise the modulename here? It is not user input, but it comes from the modules table in the database. However, if a module name would contain sequences of "../" we might be in trouble

bool function module_view(&\$theme, \$area_id, \$node_id, \$module_id) [line 385]

Function Parameters:

- *array* **&\$theme** a reference to the output object
- *int* **\$area_id** the area where we are looking for a node
- *int* **\$node_id** the node we are working with
- *int* **\$module_id** the module connected to the node we are working with

call the routine that generates the view (content) of module \$module_id

this loads the file containing the visitor interface for module \$module_id in core and subsequently calls the routine responsible for displaying the content (function modulename_view()). The routine module_view() is supposed to deposit any output into the \$theme via the appropriate methods such as \$theme->add_content().

void function update_statistics(\$node_id) [line 465]

Function Parameters:

- *int* **\$node_id** the page (node) that was viewed

update all statistics for the view of page \$node_id

this is a place for future extension. This routine is called once for every page view. It can be used to record relevant data in a table, for future reference, e.g.

- the IP-address of the visitor
- the \$node_id
- the current date/time
- the number of views of node \$node_id from the visitor's IP-address
- etc. etc.

Note that the table holding this information can quickly become very large. That requires some form of logrotate or condensing the data. This feature has yet to be developed.

- **TODO** maybe extend this routine to actually store more statistics information in a separate table

void function update_view_count(\$node_id) [line 478]

Function Parameters:

- *int \$node_id* the page (node) that need its view_count incremented with 1

update the view count for page \$node_id

manual.php

/program/manual.php - a kickstarter for the documentation

This script is also an entry point; it can be called directly. It is called from /program/admin.php via the help button. Recognised parameters:

language

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: manual.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** How about adding an extra parameter to manual.php in order to 'deep link' into the manual?
- **TODO** guess what? we need to replace this stub with real documentation
- **License** [GNU AGPLv3+Additional Terms](#)

version.php

'version.php' defines internal and external version numbers

The following constants are defined in this file:

- `WAS_VERSION` - the internal version number, e.g. 2008020100
- `WAS_RELEASE` - the external version number, e.g. 1.0 or 1.0.0
- `WAS_RELEASE_DATE` - the date that the distribution files were generated
- `WAS_ORIGINAL` - indicates the original (TRUE) or a modified version (FALSE) of this program

`WAS_VERSION` is used to see if the database version matches the program version. A difference between the two versions indicates an incomplete update. The version number is of the form `yyyymmddxx` where `yyyymmdd` is a date and the number `xx` is an auxiliary number that may or may not carry an extra meaning. `WAS_VERSION` is always greater than `WAS_VERSION` in a previous release of Website@School.

`WAS_RELEASE` is a free-format human-readable string indicating the the version of the program. It could take the form `major.minor` or `major.minor.patchlevel`.

`WAS_RELEASE_DATE` is the date on which the distribution package was generated. This date is set by editing this file `version.php` 'on the fly' from the `makedist.sh` script (see `/devel/tools/makedist.sh`).

`WAS_ORIGINAL` is a flag which indicates the original version (value TRUE) or a modified version (value FALSE) of the program. The License Agreement for Website@School states:

"In accordance with section 7(c) modified versions of the Program must clearly be marked in reasonable ways as different from the original version without misrepresenting the origin of the Program. This must be done by adding the phrase "Based on Website@School" to the Appropriate Legal Notices."

By defining `WAS_ORIGINAL` to FALSE, the phrase 'Powered by Website@School' in the interactive user interfaces will morph into 'Based on Website@School' automatically. The file `/program/about.html` should still be edited, though.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: version.php,v 1.2 2011-02-01 14:34:34 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

WAS_ORIGINAL = TRUE *[line 76]*

A boolean flag indicating this is either the original (TRUE) or a modified (FALSE) version of Website@School

WAS_RELEASE = 0.90.0 *[line 68]*

The external version number, like 1.0 or 1.0.0

WAS_RELEASE_DATE = 2011-02-01 *[line 72]*

Date of distribution file generation in ISO 8601 format: yyyy-mm-dd OR yyyy-mm-ddThh:mm:ss+0000

WAS_VERSION = 2011020100 *[line 64]*

The internal version number, like 2008012873 or 2008020100 (31 bits will work until the year 2147)

- **Used by** [error_exit\(\)](#) - indicate internal version in 'cryptic' message

Package wascore Classes

Class AclManager

[line 322]

class for manipulating (edit+save) access control lists

Overview

Every user account is associated with an access control list. This access control list boils down to a total of six tables in the database:

- acls
- acls_areas
- acls_nodes
- acls_modules
- acls_modules_areas
- acls_modules_nodes

These tables are defined as follows. acls:

acl_id serial*
permissions_jobs int
permissions_intranet int
permissions_modules int
permissions_nodes int

acls_areas:

acl_id int* (link to acls)
area_id int* (link to areas)
permissions_intranet int
permissions_modules int
permissions_nodes int

acls_nodes:

acl_id int* (link to acls)
node_id int* (link to nodes)
permissions_modules int
permissions_nodes int

acls_modules:
 acl_id int* (link to acls)
 module_id int* (link to modules)
 permissions_modules int

acls_modules_areas:
 acl_id int* (link to acls)
 module_id int* (link to modules)
 area_id int* (link to areas)
 permissions_modules int

acls_modules_nodes:
 acl_id int* (link to acls)
 module_id int* (link to modules)
 node_id int* (link to nodes)
 permissions_modules int

*marked fields are (part of) the primary key

The six tables mentioned above deal with the following permission bitmasks.

- permissions_jobs
- permissions_intranet
- permissions_modules
- permissions_nodes

The reasons to split these permission masks into six tables are:

1. Some permissions only apply to the site-level and it makes no sense to specify them for a particular combination of area, node or module. Example: permissions_jobs.
2. Some permissions can be granted for current and future objects. Example: permissions_intranet. If these permissions are granted at the site level (in table acls), then they apply not only to all current protected areas but also to all future protected areas. The same permissions could be granted on a per-area-basis but that might require adjusting the permissions once a new protected area is added to the site.
3. Sometimes it is more convenient to specify the permissions on a higher level because otherwise the size of the database may get out of hand. Example: if every user has a permission bitmask for every node on the site, the corresponding acl would have number_of_users x number_of_nodes entries. That is completely unmanageable, even for small to medium size sites.

Users and group/capacities

A user can also participate in a group in a particular capacity, e.g. member of group 'grade8' in the 'pupil'- or the 'teacher'-capacity. Every combination of group and capacity (eg

'grade8/pupil') is also associated with an access control list.

The full access control list for a user is the combination of the ACL directly associated with the user account and the ACLs associated with the group/capacities that apply to the user account. The effective permissions for a user are the result of OR'ing the permissions of all ACLs.

A specific permission is always indicated by a bit set to '1'. If a particular bit is set to '0', the user does not have the corresponding permission. This implies that the (special) bitmask 0 (zero, 32 bits are all not set) corresponds to 'no permissions at all'. It also implies that the (special) bitmask -1 (minus one, 32 bits are all set) equates to 'all permissions'.

Therefore, the **easy** way to grant access is to set the permissions bitmask to -1. This is the so-called Guru-option or -role or the Guru-permissions. However, note that granting a user or a group/capacity Guru-permissions, means that that user (these users) can do serious harm to the system because she (they) are allowed to do anything. The **safe** way is to grant as few permissions as possible.

Roles

In order to make it easier to setup the access controls and stay away from directly manipulating individual bits in a bitmask the various permission bits are combined into roles.

Two roles are always available for selection:

- permissions == 0: ROLE_NONE
- permissions == -1: ROLE_GURU

Defining other roles is done at the appropriate place, e.g. inside the code for a module.

Example: suppose that there is a module called 'Forum' which works with authenticated users. Depending on this module's permission bits the users are allowed to perform certain actions, e.g.

- read messages in the forum (bit 0, value 1)
- write messages in the forum (bit 1, value 2)
- edit their own messages (bit 2, value 4)
- edit other users' messages (bit 3, value 8)
- manage useraccounts for the forum (bit 4, value 16)

This leads to many possible combinations of set and reset bits. However, it is more practical to combine these bits into a few roles with a descriptive name:

- permissions = 1: ROLE_FORUM_VISITOR => "Visitor"
- permissions = 1+2+4 = 7: ROLE_FORUM_MEMBER => "Member"
- permissions = 1+2+4+8 = 15: ROLE_FORUM_MODERATOR => "Moderator"
- permissions = 1+2+4+8+16 = 31: ROLE_FORUM_ADMINISTRATOR => "Administrator"

By using these symbolic names for certain combinations of bitmasks it becomes easier to

manage many users and many forums without having to know what every bit means, exactly.

Obviously these roles (defined via the module in this example) will end up in a dropdown list where the appropriate role can be assigned.

Note that it is not necessary to have hierarchical roles as demonstrated in this example. It is very well possible to define two roles that must work together: `ROLE_EDITOR` could be a bitmask that allows for adding (1), editing (2), deleting (4), previewing (8) news articles whereas `ROLE_PUBLISHER` could be limited to previewing (8) and publishing (16) news articles, but not editing them. That would make sure that at least two different people are required to create and publish an article. (However, any 'Guru', with all permissions granted due to the -1 bitmask, could create + publish articles by herself.)

Module-permissions in `acls`, `acls_areas` and `acls_nodes`

The fields `permissions_modules` in the tables `acls`, `acls_areas` and `acls_nodes` should be considered as 'blanket permissions'. If a permission is set in either of these tables, the permissions apply to **all** modules at site level (`acls`), area_level (`acls_areas`) or node_level (`acls_nodes`).

Because these permissions apply to **all** modules, the only realistic roles in these cases can be either `ROLE_NONE` (permissions = 0) or `ROLE_GURU` (permissions = -1). Any other role could be meaningless for one or more modules.

Furthermore, it is a little over the top to specify permissions for **all** modules in a particular node. (It almost doesn't make sense). Therefore, the corresponding dialog only deals with these two roles `ROLE_NONE` and `ROLE_GURU` at the site level and the area level. The node level is not used for modules (but it is for pagemanager permissions - the field `permissions_nodes` - at the node level).

Typical usage

Example 1: displaying a dialog with intranet permissions for a group

```
$acl = new AclManager($output,$acl_id,ACL_TYPE_INTRANET);
$acl->set_action(array('job'=>'accountmanager','task'=>'groupsave','group'=>'8'));
$acl->set_dialog(GROUPMANAGER_DIALOG_INTRANET);
$acl->show_dialog();
```

... The result of this snippet is that a complete dialog is output to the content area of the `$output` object, including the current values from the database. The whole dialog is wrapped in a `FORM`-tag with action property based in the array set with the `set_action()` method. The dialog is POSTed with either a Save or a Cancel button.

Example 2: saving the data for the intranet permissions for a group

```
$acl = new AclManager($output,$acl_id,ACL_TYPE_INTRANET);
$acl->set_action(array('job'=>'accountmanager','task'=>'groupsave','group'=>'8'));
```

```

$acl->set_dialog($dialog);
if (!$acl->save_data()) {
    $acl->show_dialog(); // redo dialog, but without a distracting menu this time
    return;
}
...

```

The effect of this snippet is that an attempt is done to validate and save the data as it was POSTed (ie: the new values are available in `$_POST[]`). If, however, saving the data did not work, the dialog is displayed again, this time using the data from `$_POST[]` rather than from the database.

Example 3: displaying a dialog with admin permissions for a user

```

$related_acls = array($acl_id1 => "group1/capacity1", $acl_id2 => "group2/capacity2", ...);
$acl = new AclManager($this->output, $acl_id, ACL_TYPE_ADMIN);
$acl->set_related_acls($related_acls);
$acl->set_action(array('job'=>'accountmanager', 'task'=>'usersave', 'user'=>'23'));
$acl->set_dialog(USERMANAGER_DIALOG_ADMIN);
$acl->show_dialog();

```

This is comparable to example 1. The difference is that in a User-ACL there is an option to display existing permissions from the user's group/capacities. This information is displayed in the third column in the dialog. This provides a clue for the user that certain permissions might already be granted to the user via a group membership. The related permissions are communicated via an array with (integer) `acl_id`'s as key and a string value identifying the group/capacity.

- **Package** wascore
- **TODO** there is something not right with buffering the tabledefs. If an error occurs, we get FALSE instead of an array. Mmmmm....

AclManager::\$acl_id

int = 0 [line 330]

- **Var** `$acl_id` identifies the ACL we are dealing with

AcIManager::\$acl_type

int = 0 [*line 333*]

- **Var** \$acl_type identifies the type of ACL we are dealing with

AcIManager::\$area_view_areas_open

array|bool = FALSE [*line 374*]

- **Var** \$area_view_areas_open identifies which areas are currently 'open' and 'closed'

AcIManager::\$area_view_a_params

array = NULL [*line 371*]

- **Var** \$area_view_a_params holds the parameters for linking to opening/closing an area

AcIManager::\$area_view_enabled

bool = FALSE [*line 377*]

- **Var** \$area_view_enabled if TRUE we add icons to areas so they can expand/collapse (default=FALSE)

AcIManager::\$a_params_save

array|null = NULL [*line 339*]

- **Var** \$a_params_save holds the parameters for the action property of the HTML-form that is created

AcIManager::\$dialog

int = 0 [line 348]

- **Var** \$dialog identifies the exact dialog and it is added to the dialog as hidden field

AcIManager::\$dialogdef

array = NULL [line 388]

- **Var** \$dialogdef holds the current dialogdef, maybe including error messages from a failed validation

AcIManager::\$dialogdef_areas

array = array() [line 395]

- **Var** \$dialogdef_areas holds information of zero or more areas and the number of contained nodes

AcIManager::\$dialogdef_areas_total

int = NULL [line 392]

- **Var** \$dialogdef_areas_total holds the total number of items that could be displayed in the dialogdef

AcIManager::\$header

string = [line 342]

- **Var** \$header the title of the dialog, displayed at the top of the content area

AcIManager::\$intro

string = [line 345]

- **Var** \$intro the introductory text for the dialog, displayed below the \$header

AcIManager::\$output

object|null = NULL [line 327]

- **Var** collects the html output

AcIManager::\$pagination_a_params

array = NULL [line 355]

- **Var** \$pagination_a_params holds the parameters for linking to another view of the dialog

AcIManager::\$pagination_enabled

bool = FALSE [line 364]

- **Var** \$pagination_enabled if TRUE we do try to paginate the display (default=FALSE)

AcIManager::\$pagination_limit

int = NULL [line 358]

- **Var** \$pagination_limit the preferred size of a screenfull of dialog lines

AcIManager::\$pagination_offset

int = NULL [line 361]

- **Var** \$pagination_offset the record where the current screen begins

AcIManager::\$pagination_total

bool = 0 [line 384]

- **Var** \$pagination_total holds the total number of elements to display

AcIManager::\$related_acls

array = NULL [line 336]

- **Var** \$related_acls if not NULL identifies a list of acl_id => 'description' pairs with related ACLs

Constructor *void* function AcIManager::AcIManager(&\$output, \$acl_id, \$acl_type) [line 408]

Function Parameters:

- *object* **&\$output** holds the output that eventually is send to the user's browser
- *int* **\$acl_id** identifies the ACL we are dealing with (primary key in acls table)
- *int* **\$acl_type** identifies the type of ACL we are dealing with, e.g. ACL_TYPE_INTRANET or ACL_TYPE_ADMIN

constructor for the AcIManager

this constructs a new AcIManager object. Essential information such as the acl_id and the

acl_type are stored, for future reference.

array function AclManager::build_tree(\$area_id, \$acl_id, \$related_acls) [*line 1694*]

Function Parameters:

- *int* **\$area_id** the area for which to build the tree
- *int* **\$acl_id** the primary acl_id (used for both users and groups)
- *array|null* **\$related_acls** an array with related acls for this user keyed by 'acl_id' or NULL for group acls

build a tree of all nodes in an area

this routine constructs a tree-structure of all nodes in area \$area_id in much the same way as [build_tree\(\)](#) does. However, in this routine we keep the cargo limited to a minimum: the fields we retrieve from the nodes table and store in the tree are:

- node_id
- parent_id
- is_page
- title
- link_text
- module_id

Also, the tree is not cached because that does not make sense here: we only use it to construct a dialogdef and that is a one-time operation too.

int|bool function AclManager::calc_areas_total(&\$areas) [*line 896*]

Function Parameters:

- *array* **&\$areas** an array with summary information about areas, including the # of nodes to show

calculate the total number of items (site, areas, nodes) to show in dialog

the 'open' or 'closed' status of an area is dictated by \$open_areas:

- if \$open_areas is an array the elements look like \$area_id => \$show, where \$show == TRUE indicates the area is 'open' and \$show == FALSE indicates the area is 'closed'
- if \$open_area is a boolean and the value is TRUE. _all_ areas are to be considered 'open'
- otherwise _all_ areas are to be considered 'closed'.

The returned value \$total is the sum of the number of areas and the number of 'showable' nodes (as per the information in \$open_areas). If there are no areas at all, \$total is 0. If an error occurs, this routine returns FALSE.

The parameter `$areas` is used as a return value. It is keyed with `$area_id` and filled with pertinent information about the areas:

- `int $area_id`: the number of the area (also the key of the `$areas` array)
- `string $title` the name of the area
- `bool $is_active` indicating an active area (TRUE) or an inactive area (FALSE)
- `int $nodes` the total number of nodes in this area (could be 0 if no nodes were added yet)
- `int permissions_nodes` the bitmap containing the existing node permissions for this area
Also, the following information is added to the resulting array:
- `bool $show` if TRUE, all nodes in this area should be displayed
- `int $first` indicating the offset of the row for this area, relative from the start of the list of areas
- `int $last` indicating the offset of the last item to show in this area (could be the same as `$first`)

The latter three values are used to skip `$offset` rows when constructing the dialog.

- **Uses \$DB;**

array function AclManager::dialog_tableform(\$href, &\$dialogdef, [\$show_related = FALSE]) [line 1453]

Function Parameters:

- *string* **\$href** the target of the HTML form
- *array* **&\$dialogdef** the array which describes the complete dialog
- **\$show_related**

construct a form with a dialog in a table with 2 or 3 columns

this constructs a 2- or 3-column table and fills it with data from the `dialogdef`.

The first column holds the labels for the widgets. The second column holds the corresponding widget, e.g. a list box with roles. The optional third column (depends on the flag `$show_related`) shows related information. This is used to list group/capacities and roles from related groups (ie. groups of which the user is a member).

The table has headers for the columns: 'Realm', 'Role' and optional 'Related'. Rows in the table can have alternating colours via the odd/even class. This is done via the stylesheet.

- **TODO** bailing out on non-array is a crude way of error handling: this needs to be fixed

void function AclManager::enable_area_view(\$a_params, \$areas_open) [line 532]

Function Parameters:

- *array* **\$a_params** basic parameters (excluding \$area) that lead to the page where expand/collapse is processed
- *array|bool* **\$areas_open** indicator(s) for 'open' and 'closed' areas

further initialise the AclManager and enable the area expand/collapse feature

this stores the necessary information about 'open' and 'closed' areas. The parameter \$areas_open indicates the current state of affairs: (\$areas_open === FALSE) means all areas are closed (\$areas_open === TRUE) means all areas are opened If \$areas_open is an array, it contains area_id's as key and TRUE or FALSE as value. A value of TRUE indicates that an area is currently 'open', FALSE or no value set means 'closed'.

The parameters in \$a_params combined with \$WAS_SCRIPT_NAME yield an URL where the changes are processed.

void function AclManager::enable_pagination(\$a_params, \$limit, \$offset) [line 504]

Function Parameters:

- *array* **\$a_params** basic parameters (excluding \$offset and \$limit) that lead to the correct page
- *int* **\$limit** the preferred size of a screenfull of dialog lines
- *int* **\$offset** the record where the current screen begins

further initialise the AclManager and enable the dialog pagination feature

this stores the information that is necessary when a dialog has to be broken up into two or more screens (via the pagination facility in \$output). This routine stores the essential information such as the parameters that lead to the correct page (in \$a_params) and the current offset. The other necessary parameters are calculated dynamically before add_pagination() is called.

Note that pagination is only enabled after this routine is called at least once; by default we do NOT do pagination. (Actually: pagination is only used in the acl_types ACL_TYPE_PAGEMANAGER and ACL_TYPE_MODULE).

- **Uses** \$CFG;

array function AclManager::get_dialogdef_admin(\$acl_id, [\$related_acls = NULL]) [*line 1137*]

Function Parameters:

- *int* **\$acl_id** the acl of interest
- *array* **\$related_acls** NULL or a list of acl_id => "group/capacity" pairs for related permissions

construct an array with the admin dialog information

this creates an array with widgets for all possible admin jobs for \$acl_id.

This dialog is supposed to be rendered as a 2-column (group acl) or 3 column (user acl) table. The contents of the 3rd column is a list (an array) of related permissions, ie. the permissions a user has been granted via a group membership. The related information is stored in an extra array element 'related'.

The related information is constructed only in the case where \$related_acls is not NULL.

The dialog is filled with the current values via \$item['value'] but as a side effect the current value is also recorded in \$item['old_value']). This makes it easier to determine whether any values have changed (see [save_data_admin\(\)](#)).

- **TODO** handle the related information in this dialog

array function AclManager::get_dialogdef_intranet(\$acl_id, [\$related_acls = NULL]) [*line 1020*]

Function Parameters:

- *int* **\$acl_id** the acl of interest
- *array* **\$related_acls** NULL or a list of acl_id => "group/capacity" pairs for related permissions

construct an array with the intranet dialog information

this creates an array with 1 or more list boxes with the current roles for \$acl_id for intranet access at the site level and for individual private areas. This dialog is supposed to be rendered as a 2-column (group acl) or 3 column (user acl) table. The contents of the 3rd column is a list (an array) of related permissions, ie. the permissions a user has been granted via a group membership. The related information is stored in an extra array element 'related'.

The related information is constructed only in the case where \$related_acls is not NULL.

The dialog is filled with the current values via \$item['value'] but as a side effect the current value is also recorded in \$item['old_value']). This makes it easier to determine whether any values have changed (see save_data_internet()).

- **TODO** handle the related information in this dialog

void function AclManager::get_dialogdef_pagemanager(\$acl_id, \$related_acls) [*line 1230*]

Function Parameters:

- **\$acl_id**
- **\$related_acls**

construct a dialog definition for pagemanager permissions

string function AclManager::get_icon_area(\$area_id, \$area_is_open, \$offset) [*line 1873*]

Function Parameters:

- *int* **\$area_id** the area to open/close (0 means: open site level)
- *bool* **\$area_is_open** current status
- *int* **\$offset** the position of this icon in the current list of items

construct a clickable icon to open/close this area

This is a toggle: if the area is closed the closed icon is shown, but the action in the A-tag is to open the icon (and vice versa).

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

string function AclManager::get_icon_blank() [*line 1913*]

construct a spacer of standard icon width (to line up items)

- **Uses** \$USER
- **Uses** \$CFG

array function AclManager::get_permissions(\$acl_id, [\$related_acls = NULL]) [*line 1546*]

Function Parameters:

- *int* **\$acl_id** the primary acl_id (used for both users and groups)
- *array|null* **\$related_acls** an array with related acls for this user or NULL for group acls

retrieve an array with 0, 1 or more records with permissions from table 'acls'

this constructs an array with all (or selected) permissions from the 'acls' table for the specified acl \$acl_id and optionally for all related acl_id's in \$related_acls. The resulting array is keyed by acl_id.

array function AclManager::get_permissions_areas(\$acl_id, [\$related_acls = NULL], [\$areas = NULL]) [*line 1580*]

Function Parameters:

- *int* **\$acl_id** the primary acl_id (used for both users and groups)
- *array|null* **\$related_acls** an array with related acls for this user keyed by 'acl_id' or NULL for group acls

- *array|null* **\$areas** an array with areas of interest keyed by 'area_id' or NULL for all areas

retrieve an array with 0, 1 or more records with permissions from table 'acls_areas'

this constructs an array with all permissions from the 'acls_areas' table for the specified acl \$acl_id and optionally for all related acl_id's in \$related_acls and optional areas. The resulting array is keyed by area_id and acl_id.

Note that by making the result keyed by area_id first (and then acl_id) it becomes possible to step through a list of areas and have 0,1 or more acls for that area in a single array, e.g. \$acls = \$permissions[16] yields the selected acls that apply to area 16. That is handy when constructing dialogs iterating through areas such as intranet permissions.

array function AclManager::get_permissions_nodes_in_area(\$area_id, \$acl_id, [\$related_acls = NULL]) [*line 1635*]

Function Parameters:

- *array* **\$area_id** the area where the nodes reside
- *int* **\$acl_id** the primary acl_id (used for both users and groups)
- *array|null* **\$related_acls** an array with related acls for this user keyed by 'acl_id' or NULL for group acls

retrieve an array with 0, 1 or more records with permissions from table 'acls_nodes'

this constructs an array with all permissions from the 'acls_nodes' table for the specified acl \$acl_id and optionally for all related acl_id's in \$related_acls and optional nodes. The resulting array is keyed by node_id and acl_id.

Note that by making the result keyed by node_id first (and then acl_id) it becomes possible to step through a list of nodes and have 0,1 or more acls for that node in a single array, e.g. \$acls = \$permissions[16] yields the selected acls that apply to node 16. That is handy when constructing dialogs iterating through nodes such as pagemanager permissions.

array function AclManager::get_roles_intranet() [*line 1372*]

construct an option list with roles for intranet access

array function AclManager::get_roles_pagemanager([\$level = ACL_LEVEL_NONE]) [*line 1390*]

Function Parameters:

- *int* **\$level** limits permissions to level 'page', 'section', 'area' or 'site'

construct an option list with roles for pagemanager access

bool function AclManager::save_data() [line 579]

save the changed data for the selected acl_type

this interprets the data from the selected dialog and saves the (changed) permission data accordingly. This, too, is merely a dispatcher to the subroutines that do the actual work.

bool function AclManager::save_data_admin() [line 763]

save changed job permissions to the database

this saves the changed job permissions to the acls table.

If the user selected the guru option, we simply set the permissions to JOB_PERMISSION_GURU (i.e. all permissions set). If not, we iterate through all existing permissions and set the corresponding bits. After that the data is saved to the correct acls-record.

- **TODO** fix the crude error check on dialogdef === FALSE here

bool function AclManager::save_data_intranet() [line 640]

save the changed roles for intranet access to the tables 'acls' and 'acls_areas'

this interprets the data from the intranet dialog and saves the changed roles accordingly

bool function AclManager::save_data_pagemanager() [line 983]

save the changed roles for pagemanager to the tables 'acls' and 'acls_areas' and 'acls_nodes'

this interprets the data from the pagemanager dialog and saves the changed roles accordingly

bool function AclManager::save_data_permissions() [line 655]

save the changed roles in the dialog to the corresponding tables 'acls'

this interprets the data from the current dialog and saves the changed roles accordingly. Note that the information about tables and fields etc. is all contained in the dialogdef so we can use this generic save_data() routine.

void function AclManager::set_action([\$a_params = NULL]) [line 439]

Function Parameters:

- *array \$a_params*

further initialise the AclManager with the dialog action property

this stores an array with parameters that must be added to the action property of the HTML form that will be POSTed, i.e. the URL to which the dialog will be posted. Example of such an array is: `array('job' => 'accountmanager', 'task' => 'user_save', 'user' => 123);` The information in this array is later combined with `WAS_SCRIPT_NAME`.

void function AclManager::set_dialog([\$dialog = 0]) [line 480]

Function Parameters:

- *int \$dialog* a unique identification (within this job) of the dialog

further initialise the AclManager with the dialog identification

this stores an integer number that is used to identify the dialog. This number is subsequently added to the dialog as a hidden field, which makes it possible to identify the dialog once it is POSTed

void function AclManager::set_header([\$header = ""]) [line 453]

Function Parameters:

- *string \$header* text to show as title

further initialise the AclManager with the dialog header

this stores a string that is used as a title for the dialog Note that this header may be extended with a (translated) string like `'[{FIRST}]-{LAST} of {TOTAL}]'` in case of a paginated display.

void function AclManager::set_intro([\$intro = ""]) [line 466]

Function Parameters:

- *string \$intro* introductory text for the dialog

further initialise the AclManager with the dialog introductory text

this stores a string that is displayed after the dialog header. This text supposedly contains some more information about the dialog.

void function AclManager::set_related_acls([\$related_acls = NULL]) [line 423]

Function Parameters:

- **array \$related_acls** identifies a list with related ACLs

further initialise the AclManager with related Acl's

this stores the array with 0, 1 or more key-value-pairs of the form \$acl_id => \$group_capacity_name, e.g. 3 => 'staff/member', 4 => 'grade7/teacher'

void function AclManager::show_dialog() [line 547]

show the dialog where the selected Acl can be modified

this shows the dialog corresponding to the acl_type that was previously selected, including existing data from the previously selected acl_id. Note that this routine is only a simple dispatcher; actual work is done in subroutines.

void function AclManager::show_dialog_admin() [line 738]

display a tabular form for manipulating admin permissions

This dialog is a table consisting of 2 (group acl) or 3 (user acl) columns. The first column holds the various job names/descriptions. The second column holds a checkbox for the job. The optional third column holds corresponding (existing) permissions based on a group/capacity membership of the user. This 3rd column is displayed only when there are related acls (indicated via related_acls not empty)

void function AclManager::show_dialog_intranet() [line 620]

display a tabular form for manipulating intranet permissions

This dialog is a table consisting of 2 (group acl) or 3 (user acl) columns. The first column holds the text 'All areas' or the name of a private area (if any) The second column holds a listbox where the user can select 1 out of 3 roles: 0 = "--", 1 = "Access", -1 = "Guru". The optional third column holds corresponding (existing) roles based on a group/capacity membership of the user. This 3rd column is displayed only when there are related acls (indicated via related_acls not empty)

void function AclManager::show_dialog_pagemanager() [line 836]

display a tabular form for manipulating pagemanager permissions

This dialog is a table consisting of 2 (group acl) or 3 (user acl) columns. The first column identifies the site, areas or nodes within areas. The second column holds a listbox where the user can select a role for that particular item. The roles 0 = "--" and -1 = "Guru" are always available. The optional third column holds corresponding (existing) roles based on a group/capacity membership of the user. This 3rd column is displayed only when there are related acls (indicated by \$related_acls not being empty).

The main purpose of this routine is to show some \$this->pagination_limit table rows (starting at \$this->pagination_offset) and corresponding [Save] and [Cancel] buttons that eventually lead to the save routine. (If pagination is not enabled, the full overview is displayed).

Note that the actual pagination is performed in [get_dialogdef_pagemanager\(\)](#). The additional feature of expanding/collapsing areas in the display is also done in [get_dialogdef_pagemanager\(\)](#).

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG;

void function AclManager::show_tree_walk(&\$dialogdef, &\$tree, &\$permissions_nodes, &\$index, \$node_id, \$first, \$last, \$acl_id, &\$related_acls) [line 1786]

Function Parameters:

- *array* **&\$dialogdef** collects the widgets
- *array* **&\$tree** a reference to the complete tree built earlier
- *array* **&\$permissions_nodes** contains permissions per node
- *int* **&\$index** only add node to dialogdef if \$index is between \$first and \$last, increments for every node
- *int* **\$node_id** the first node of this tree level to show
- *int* **\$first** lower bound of interval
- *int* **\$last** upper bound of interval
- *int* **\$acl_id** the acl we are rendering

- *array|null* **related_acls** an array with related acs for this user keyed by 'acl_id' or NULL for group acs

add the specified node to dialogdef, optionally all subtrees, and subsequently all siblings

this routine adds a widget to the dialogdef for the specified node After that, any subtrees of this node are added too, using recursion This continues for all siblings of the specified node until there are no more (indicated by a sibling_id equal to zero).

- **Usedby** [AclManager::show_tree_walk\(\)](#)
- **Uses** [AclManager::show_tree_walk\(\)](#)
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

Class AdminOutput

[line 549]

conveniently collect output

This class allows for a convenient way to temporarily store output, in random order and still being able to output to the browser in the correct order (eg. headers() first, etc.).

This class 'knows' everything about the structure of a generated page. Most of this knowledge is contained in \$this->get_html(); The actual layout is defined in the corresponding stylesheets, e.g. admin_base.css.

Typical use of this object is to add HTML-code to various parts of the page via the add_*() methods and finally sending the collected output to the user's browser with \$this->send_output(). That's it.

- **Package** wascore
- **TODO** carefully check if we need more headers in html-head section of document, see [AdminOutput\(\)](#).
- **TODO** add a 'funnel mode': disable all distracting links that could seduce the user to leave and leave locked records (eg. nodes)

AdminOutput::\$breadcrumbs

```
array = array() [line 587]
```

- **Var** a list URL components etc. identifying the path that leads to the current screen

AdminOutput::\$content

```
array = array() [line 569]
```

- **Var** collection of items/lines that are part of the content area

AdminOutput::\$dtd

```
string = <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"> [line 551]
```

- **Var** the standard doctype (default: HTML 4.01 Transitional)

AdminOutput::\$funnel_mode

```
bool = FALSE [line 593]
```

- **Var** if TRUE, the display should be in funnel-mode, ie. distracting links should be non-working

AdminOutput::\$helptopic

string = [line 584]

- **Var** the additional parameter to add to the help link in the navigation

AdminOutput::\$high_visibility

bool = FALSE [line 581]

- **Var** this switches the navigation between image-based and text-based

AdminOutput::\$html_head

array = array() [line 563]

- **Var** collection of items/lines that will be output as part of the HTML-head section

AdminOutput::\$http_headers

array = array() [line 560]

- **Var** collection of individual http-headers that are to be sent *_before_* any HTML is sent

AdminOutput::\$menu

array = array() [line 566]

- **Var** collection of items/lines that are part of the menu (could be empty)

AdminOutput::\$messages_bottom

array = array() [line 578]

- **Var** collection of messages that are to be displayed via a javascript alert() at END of page

AdminOutput::\$messages_inline

array = array() [line 572]

- **Var** collection of messages that are to be displayed just below the navigation bar

AdminOutput::\$messages_top

array = array() [line 575]

- **Var** collection of messages that are to be displayed via a javascript alert() at START of page

AdminOutput::\$pagination

array = array() [line 590]

- **Var** a list URL components etc. comprising a list of links to paginated display of lists

AdminOutput::\$subtitle

string = [line 557]

- **Var** a subtitle displayed in the generated page (for the time being it is empty or has a)

AdminOutput::\$suppress_output

bool = FALSE [line 596]

- **Var** if TRUE, no output should be sent whatsoever

AdminOutput::\$title

string = [line 554]

- **Var** the title to display in both the title tag and in the page itself (usually the sitename)

Constructor *void* function AdminOutput::AdminOutput([\$title = "], [\$subtitle = "]) [line 610]

Function Parameters:

- *string* **\$title** the title to display in both the title tag and inside the document
- *string* **\$subtitle** the text to display underneath the title in the document (or if empty)

constructor

This sets up the object and adds the title and subtitle.

- **TODO** is it really wise to add a base header? It interferes with the session cookie whenever you login at another URL than the base+'admin.php'... Comment it out for now
- **TODO** do we need a link rel="shortcut icon" type of header too?
- **TODO** do we really need more meta-headers?

void function AdminOutput::add_breadcrumb(\$href, [\$params = NULL], [\$attributes = NULL], [\$anchor = NULL])

[line 1452]

Function Parameters:

- *string* **\$href** holds the hypertext reference
- *string|array* **\$params** holds the parameters to add to the \$href
- *string|array* **\$attributes** holds the attributes to add to the tag
- *string* **\$anchor** the text that identifies the breadcrumb

add a breadcrumb to the breadcrumb trail

this stores information about a crumb in the breadcrumb trail into the breadcrumbs array. We store this information in pieces that are readily usable with [html_a\(\)](#). However, by constructing the links at the latest possible time, we are able to suppress the real links, by replacing the href + parameters with a bare "#". This trick can be used to keep the user focussed on the task at hand (in 'funnel-mode').

void function AdminOutput::add_content(\$content) [line 1409]

Function Parameters:

- *string|array* **\$content** the line(s) of text to add

add a line or array of lines to the content part of the document

void function AdminOutput::add_html_header(\$headerline) [line 1323]

Function Parameters:

- *string* **\$headerline** headerline to add

add a header to the HTML head part of the document

void function AdminOutput::add_http_header(\$headerline) [line 1314]

Function Parameters:

- *string* **\$headerline** headerline to add

add an HTTP-header

void function AdminOutput::add_menu(\$menuline) [line 1422]

Function Parameters:

- *string* **\$menuline** the line of text to add

add a line to the menu part of the document

void function AdminOutput::add_message(\$message) [line 1361]

Function Parameters:

- *string/array* **\$message** message(s) to add inline

add a message to the list of inline messages, part of the BODY of the document

void function AdminOutput::add_meta(\$meta) [line 1386]

Function Parameters:

- *array* **\$meta** an array with name-value-pairs that should be added to the HTML head part

add a line with meta-information to the HTML head part of the document

void function AdminOutput::add_meta_http_equiv(\$meta) [line 1398]

Function Parameters:

- *array* **\$meta** an array with name-value-pairs that should be added to the HTML head part

add a line with http-equiv meta-information to the HTML head part of the document

int function AdminOutput::add_pagination(\$href, \$base_params, \$num_records, \$limit, \$current_offset, \$num_links) [line 1510]

Function Parameters:

- *string* **\$href** the script to call

- *array* **\$base_params** the necessary parameters to land on the correct page
- *int* **\$num_records** the total length of the list
- *int* **\$limit** the preferred size of the screen to show
- *int* **\$current_offset** the record that starts the current screen
- *int* **\$num_links** the maximum number of pages to show in the navbar (excluding Prev/Next/All)

add a pagination navigation bar to the output

this adds a pagination navbar to the output making it easier to step through a long listing of items a screen at a time

Features:

- the 'Prev' and 'Next' buttons do wrap: whenever we hit the begin/end of the list, we start again at the end/begin.
- if there are more screens to show than \$num_links, we show at most \$num_links links whenever we are NOT at the start of the list, the smallest link is displayed differently (via translation), e.g. via a left bracket '<' or three dots '...'
- the same trick is used at the end of the list to indicate there's more (via a different translation), e.g. via a right bracket '>' or three dots '...'

This function returns the number of screens that is required to show all the items in screens of at most \$limit items. The number of screens is at least 1.

void function AdminOutput::add_pagination_item(\$href, [\$params = NULL], [\$attributes = NULL], [\$anchor = NULL]) [*line 1476*]

Function Parameters:

- *string* **\$href** holds the hypertext reference
- *string|array* **\$params** holds the parameters to add to the \$href
- *string|array* **\$attributes** holds the attributes to add to the tag
- *string* **\$anchor** the text that identifies the link

add a link to screen of a paginated list to the existing list

this stores information about a screen in the paginated display of a list. We store this information in pieces that are readily usable with [html_a\(\)](#). However, by constructing the links at the latest possible time, we are able to suppress the real links, by replacing the href + parameters with a bare "#". This trick can be used to keep the user focussed on the task at hand (in 'funnel-mode').

void function AdminOutput::add_popup_bottom(\$message) [line 1347]

Function Parameters:

- *string|array* **\$message** message(s) to add

add a message to the list of popup-messages at the BOTTOM of the document

void function AdminOutput::add_popup_top(\$message) [line 1333]

Function Parameters:

- *string|array* **\$message** message(s) to add

add a message to the list of popup-messages at the TOP of the document

void function AdminOutput::add_stylesheet(\$url) [line 1375]

Function Parameters:

- *string* **\$url** url of the stylesheet

add a link to a stylesheet to the HTML head part of the document

string function AdminOutput::get_address([\$m = ""]) [line 1227]

Function Parameters:

- *string* **\$m** left margin for increased readability

return the reconstructed URL in a single (indented) line

This constructs the exact URL (including the GET-parameters) of the current script. This URL is returned as HTML so it can be displayed. It is NOT meant to be a clickable link, but as a documentation of the actual URL that was used. Note that this URL can be suppressed by an appropriate 'display:none' in the stylesheet, making it an item that only appears on a hardcopy (media="print") and not on screen.

string function AdminOutput::get_bottomline([\$m = ""]) [line 1201]

Function Parameters:

- *string \$m* left margin for increased readability

report basic performance indicators in a single line

This calculates the execution time of the script and the number of queries. Note a special trick: we retrieve the translated string in a dummy variable before calculating the number of queries because otherwise we might miss one or more query from the language/translation subsystem.

Note that the message containing the performance indicators is only generated when debug is TRUE; the information is not that interesting for ordinary users.

string function AdminOutput::get_breadcrumbs([\$m = ""]) [*line 1250*]

Function Parameters:

- *string \$m* left margin for increased readability

retrieve/construct a list of 0 or more clickable breadcrumbs

this reads the breadcrumbs-array and constructs the breadcrumb trail. If \$this->funnel_mode is TRUE, the links are still clickable but the href-part is replaced with a "#" instead of the 'real' href + parameters.

string function AdminOutput::get_content([\$m = ""]) [*line 778*]

Function Parameters:

- *string \$m* left margin for increased readability

get all lines in the content DIV in a single properly indented string

string function AdminOutput::get_div_messages([\$m = ""]) [*line 827*]

Function Parameters:

- *string \$m* left margin for increased readability

get a perhaps bulleted list of messages in a DIV

This constructs an unordered list with messages, if there are any. If there is no message at all, an empty string is returned (without DIV). If there is a single message, no bullet is added to the message. If there are two or more messages, bullets are added.

Note that this routine is an exception with respect to the DIV-tags: this helper routine DOES generate its own DIVs whenever there is at least 1 message. This means that there is no DIV at all when there are no messages.

string function AdminOutput::get_html() [line 705]

construct an output page in HTML

This constructs a full HTML-page, starting at the DTD and ending with the html closing tag.

The page is constructed using nested DIVs, the layout is taken care of in a separate style sheet. All knowledge about the structure of the page is contained in this routine.

The performance of the script (# of queries, execution time) is calculated as late as possible, to catch as much as we can. Therefore the construction is done in two parts and performance is calculated last.

The contents of the various DIVs is constructed in various helper routines in order to make this routine easy to read (by humans that is). The various helper routines all are called with a string of space characters; this should improve the readability of the page that is generated eventually.

Note that the routine `$this->get_div_messages()` does in fact generate its own DIV tags. This is done in order to completely get rid of the message DIV, we do not even want to see an empty DIV if there are no message.

string function AdminOutput::get_html_head([\$m = ""]) [line 768]

Function Parameters:

- *string \$m* left margin for increased readability

get all lines in the HTML head section in a single properly indented string

string function AdminOutput::get_lines(\$a, [\$m = ""]) [line 801]

Function Parameters:

- *string \$m* left margin for increased readability

- **\$a**

get lines from an array in a single properly indented string

This is a workhorse to convert an array of lines to a properly indented block of text.

string function AdminOutput::get_logo([\$m = ""]) [*line 895*]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct an image tag with the W

string function AdminOutput::get_menu([\$m = ""]) [*line 788*]

Function Parameters:

- *string* **\$m** left margin for increased readability

get all lines in the menu DIV in a single properly indented string

string function AdminOutput::get_navigation([\$m = "], [\$textonly = FALSE]) [*line 1027*]

Function Parameters:

- *string* **\$m** left margin for increased readability
- *bool* **\$textonly** if TRUE, no images are used to construct navigation links

construct a navigation bar for various jobs the user is allowed to do

This constructs an unordered list (UL) with items, where each item is a clickable link to one of the various 'managers' (e.g. page manger, file manager, etc.).

There are several different 'modes' for this function. The type of output not only depends on the \$textonly flag, but also on the user's permissions for the various 'managers'. This information is retrieved from the global \$USER object.

The list of available options is stored in an array of arrays. This should make it easier to add a new 'manager' in the future: simply add another element to the \$items array below (and of course the necessary changes in the dispatcher in main_admin() above).

As a rule the links are presented to the user in the form of clickable images (icons) of 32x32 pixels. If no image is specified, a text-link is used instead. If the optional flag `$textonly` is set, `_all_` links are displayed as a text-link.

Depending on the user's privileges, access to some 'managers' is denied. This is visualised by displaying either a black/white 32x32 image (instead of the coloured icon) or by adding the class 'dimmed' to the text-based anchor tags. This makes it possible to show 'dimmed' or 'greyed out' text if the user has no access. Note, however, that these 'forbidden' links are not suppressed: the W@S philosophy says that everything should be transparent as possible and that rules out the option to suppress the things the user is not supposed to do. If the user does follow the 'dimmed' links, an error message is displayed (see the code in the dispatcher in [main_admin\(\)](#) above). One last note on this issue of denying access in a transparent way: the mousover already indicate that access will be denied. That should provide an extra clue to the user.

The list of items contains a link to the start centre (the first link) and also a link to the documentation (the last link). Both items are governed by the same privilege mask: `JOB_PERMISSION_STARTCENTER`. Basically it means that everyone with minimal administrator privileges has access to both the start centre and the help function. Effectively this is everyone that has access to `admin.php` so the images `startcenter-bw.gif` and `help-bw.gif` do not really make sense, but for completeness sake...

- **TODO** we need to clean up this code and properly implement the funnel mode (2009-12-18)

string function AdminOutput::get_pagination([\$m = ""]) [line 1285]

Function Parameters:

- *string* **\$m** left margin for increased readability

retrieve/construct a list of 0 or more clickable links to paginated screens

this reads the pagination-array and constructs a list of links to individual screens of a paginated display of a list.

If `$this->funnel_mode` is `TRUE`, the links are still clickable but the href-part is replaced with a `"#"` instead of the 'real' href + parameters.

string function AdminOutput::get_popups(\$messages, [\$m = ""]) [line 862]

Function Parameters:

- *string* **\$m** left margin for increased readability
- *array* **\$messages** @messages a collection of message to display via alert()

construct javascript alerts for messages

This constructs a piece of HTML that yields 0 or more calles to the javascript alert() function, once per message. If no messages need to be displayed an empty string is returned.

string function AdminOutput::get_quickbottom([\$m = ""]) [line 970]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct a list of quicklinks for botton of page

This creates HTML-code for a few links that can be displayed at the bottom of the page. Currently this list is has 1 link. Possible links are

- a logout link (which will end the user's session)
- a link to the main site (i.e. /index.php without any addition parameters)
- a link to the version checker on the project's home page
(see also [get_quicktop\(\)](#)).

string function AdminOutput::get_quicktop([\$m = ""]) [line 926]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct a list of quicklinks for top of page, including logout link

This creates HTML-code for a few links that can be displayed at the top of the page. These links are:

- a logout link (which will end the user's session)
- a link to the main site (i.e. /index.php without any addition parameters)
- a link to the version checker on the project's home page

Note that the latter opens in a smallish new window that can be easily dismissed without 'losing' the main page.

void function AdminOutput::send_headers() [line 641]

send collected HTTP-headers to user's browser

This sends the headers that still need to be sent. These are collected in the array `$this->http_headers`. If headers are already sent, this fact is logged (and the collected headers are not sent).

void function AdminOutput::send_output() [line 670]

send collected output to user's browser

This first sends any pending HTTP-headers and subsequently outputs the page that is constructed by `$this->get_html()`. However, if the flag `suppress_output` is set then nothing is sent, not even a header. This allows for routines that interact directly with the user, e.g. a function to download a file.

bool function AdminOutput::set_funnel_mode(\$funnel_mode) [line 1585]

Function Parameters:

- *bool* **\$funnel_mode** if TRUE, funnel mode is switched on and distracting navigation links rendered inactive

manipulate the funnel mode

string function AdminOutput::set_helptopic(\$topic) [line 1431]

Function Parameters:

- *string* **\$topic** the topic in the help function

set the additional help topic to show when user clicks help button

bool function AdminOutput::set_suppress_output(\$suppress_output, \$suppress) [line 1597]

Function Parameters:

- *bool* **\$suppress** if TRUE, the actual output is suppressed (effectively disables `send_output()`)
- **\$suppress_output**

Class Area

[line 42]

Methods to access properties of an area

- **Package** wascore
- **TODO** refactor/change the way the default node and area are calculated from the requested node and area. We now go the the database too often.

Area::\$area_exists

bool = FALSE [line 65]

- **Var** FALSE if area uninitialised/not found, otherwise TRUE

Area::\$area_id

int = NULL [line 50]

- **Var** the actual area id, calculated from \$requested_area and \$requested_node

Area::\$area_record

array = FALSE [line 62]

- **Var** cached area record from database

Area::\$module

the = NULL [line 75]

- **Var** module that actually provides the content

Area::\$nodes

array = array() [line 56]

- **Var** list of cached node records keyed with node_id

Area::\$node_id

int = NULL [line 53]

- **Var** the actual node id, calculated from \$requested_area and \$requested_node

Area::\$requested_area

int|null = NULL [line 44]

- **Var** the requested area or NULL if not specified

Area::\$requested_node

int|null = NULL [line 47]

- **Var** the requested node or NULL if not specified

Area::\$table_areas_prefix

string = [line 68]

- **Var** the name of the areas table including table prefix

Area::\$table_nodes_prefix

string = [line 71]

- **Var** the name of the nodes table including table prefix

Area::\$tree

array = array() [line 59]

- **Var** list of arrays acting as a tree structure of all nodes in the area

Constructor *void* function Area::Area([\$requested_area = NULL], [\$requested_node = NULL], \$area_id, \$node_id) [line 91]

Function Parameters:

- *int|null* **\$area_id** the number of the area the user requested (or NULL if none specified)
- *int|null* **\$node_id** the number of the node to display (or NULL if none specified)
- **\$requested_area**
- **\$requested_node**

construct an Area object

This initialises the Area by calculating the correct node_id (of a page to show) and area_id. If all goes well, initialisation continues with retrieving `_all_` node records in an (ordered) array `$this->nodes` and subsequently constructing the complete navigation tree from the retrieved node records, taking 'hidden' pages/sections into account. If all is well, the module referenced by the node `$node_id` is initialised. Success is indicated by setting the variable `$this->exists` to TRUE.

- **Uses \$DB**

bool function Area::area_is_private() [line 150]
determine if an area is private or public

bool function Area::build_tree_of_nodes(&\$records, \$records) [line 664]
Function Parameters:

- **array \$records** pointer to ordered array of node records (pointer to save stack space)
- **&\$records**

construct a complete tree from node records (including 'hidden' nodes)

This iterates through all node records `$records` and constructs a tree. Nodes/sections without children are automatically set to 'hidden'. If all children of a node/section are hidden, that node/section is also hidden.

The resulting tree is actually an array keyed with the node_id (and with an additional 'node' 0, see notes below). The order of the array is not relevant because any node can be accessed directly via the key. However, the order is the same as the order of the incoming `$records` array but with 'node' 0 prepended.

Note 1 It is essential that the incoming array `$record` is properly ordered, i.e. all nodes first grouped by parent_id and after that sorted by sort_order. This way it is possible to easily construct a linked list of siblings in the correct sort order.

Note 2 The root node of the whole tree has node_id 0. The value 0 is also used to indicate the end of a linked list. Note that there is no node with node_id 0 in the database; the first node has node_id = 1 so there are no conflicts here.

Note 3 The root node in the tree structure has node_id 0. In order to have the top level nodes (ie. nodes directly under the root node) have their parent_id's set to 0 instead of to their own node_id. The reason to use parent_id = node_id is that referential integrity requires that a the

parent_id field should have a valid node_id and 0 is not a valid node_id in the database.

bool/int function Area::calculate_default_descendant_node_id(\$area_id, \$subtree_id, &\$node_cache, \$node_cache) [line 562]

Function Parameters:

- *int* **\$area_id** a valid indicator of the area to look at
- *int|null* **\$subtree_id** the id of the starting node of the subtree to search OR null if searching whole area
- *array* **\$node_cache** contains cached records, used to identify circular reference
- **&\$node_cache**

calculate the default page in the subtree \$subtree_id in area \$area_id

This searches for the first default page in the subtree starting at node (of type section) \$subtree_id. If \$subtree_id is empty, the whole area is searched for a default page.

- **Uses** MAXIMUM_ITERATIONS - limits the number of levels to try to a sensible maximum (no endless loops)

bool/int function Area::calculate_node_id(\$requested_area, \$requested_node, &\$node_cache) [line 376]

Function Parameters:

- *\$requested_area* **\$requested_area** integer|null the requested node number or null if none specified
- *\$requested_node* **\$requested_node** integer|null the requested area number or null if none specified
- *\$node_cache* **&\$node_cache** array this referenced variable holds the node records retrieved from the database

determine which node in which area to show

bool/int function Area::calculate_validate_default_node_id(\$requested_area, &\$node_cache) [line 400]

Function Parameters:

- *\$requested_area* **\$requested_area** integer|null the requested area number or null if none specified
- *\$node_cache* &**\$node_cache** array this referenced variable holds the node records retrieved from the database

calculate and validate the default node from an area or the default area

This determines the default node in the specified area or in the default area if no area was explicitly requested. Returns the node_id OR FALSE if no suitable node can be found.

As a side-effect the node-records that are retrieved from the database in the process are cached in \$node_cache, with the corresponding node_id as the key.

bool/int function Area::calculate_validate_node_id(\$requested_area, \$requested_node, &\$node_cache) [*line 455*]

Function Parameters:

- *\$requested_area* **\$requested_area** integer|null the requested node number or null if none specified
- *\$requested_node* **\$requested_node** integer|null the requested area number or null if none specified
- *\$node_cache* &**\$node_cache** array this referenced variable holds the node records retrieved from the database

calculate and validate the node to display based on a node and an area or the default area

This determines whether the specified node (and area if specified) is a valid node. If the specified node is not a page (but a section), we descend into the subtree starting at that node until we find a (default) node that is also a page.

If no suitable node can be found, this routine returns FALSE. Otherwise the valid node_id of the page is returned, even if the requested node was a section. In other words: this routine may return another node_id than the one that was requested.

bool function Area::exists() [*line 132*]

determine existence of area

string function Area::get_area_title() [*line 164*]

fetch the title to be used in a HTML-title-tag in the head section

This tries to retrieve the area title. If that title is not defined (i.e. empty), the global title of the site is used.

- **Uses \$CFG**

bool|array function Area::get_breadcrumb_anchors([\$node_id = NULL], [\$attributes = NULL]) [*line 240*]

Function Parameters:

- *int* **\$node_id** indicates which node, NULL means the default node
- *array|null* **\$attributes** NULL or an array of key-value-pairs to add to the anchors

fetch breadcrumb trail for a node

array function Area::get_children(\$node_id, [\$show_hidden_too = FALSE]) [*line 275*]

Function Parameters:

- *int* **\$node_id** the direct parent of the nodes that will be returned
- *int* **\$show_hidden_too** if TRUE also the 'hidden' children are returned

get an ordered array of node records with parent equal to \$node_id

This returns an ordered array of all children (pages and sections) of node \$node_id. If \$node_id is itself a page, then an empty array is returned. Note that only the direct descendants are returned, not the grandchildren.

string function Area::get_node_link_href([\$node_id = NULL]) [*line 297*]

Function Parameters:

- *int|null* **\$node_id** indicates the node to look at or the default node if NULL

get the link_href property of a node

This retrieves the link_href property of the specified \$node_id or the default node.

bool|array function Area::get_node_record([\$node_id = NULL]) [*line 203*]

Function Parameters:

- *int* **\$node_id** indicates the node record to retrieve

get a node record, maybe from the cache

This retrieves a node record, either from the cache or otherwise from the database. If data is retrieved from the database it is added to the cache. The possible embargo is taken into account.

bool|string function Area::get_node_title([\$node_id = NULL]) [*line 183*]

Function Parameters:

- *int* **\$node_id** indicates which node, NULL implies the default node

fetch the title of a node

This tries to retrieve the title of the node. The result could be an empty string; there is no 'built-in' title such as 'page \$node_id' or something.

int function Area::get_theme_id() [*line 141*]

determine the theme to use

string function Area::node2anchor(\$node_record, [\$attributes = NULL], [\$textonly = FALSE]) [*line 328*]

Function Parameters:

- *array* **\$node_record** the node record to convert
- *array* **\$attributes** optional attributes to add to the HTML A-tag
- *bool* **\$textonly** if TRUE, no clickable images will be returned

construct an anchor from a node record

This constructs an array with key-value-pairs that can be used to construct an HTML anchor tag. At least the following keys are created in the resulting array: 'href', 'title' and

'anchor'. The latter is either the text or a referent to an image that is supposed to go between the opening tag and closing tag. Furthermore an optional key is created: target. The contents of the input array \$attributes is merged into the result.

If the parameter \$textonly is TRUE the key 'anchor' is always text. If \$textonly is NOT TRUE, the 'anchor' may refer to an image.

Note that the link text is always non-empty. If the node record has an empty link_text, the word 'node' followed by the node_id is returned. (Otherwise it will be hard to make an actual clickable link).

void function Area::retrieve_nodes_from_database(\$area_id) [line 623]

Function Parameters:

- *int \$area_id* the area for which all nodes are to be retrieved

get an array of all available node records in the selected area as assoc arrays

This yields an array of all available (i.e. visible and hidden) nodes in the area. The expired nodes and nodes under embargo are not retrieved; these are considered non-existing.

The array with nodes is ordered by parent_id and sort_order and keyed with the node_id. This sort order helps to create an ordered list of nodes per level.

bool function Area::subtree_is_hidden(\$node_id, &\$tree, \$tree) [line 748]

Function Parameters:

- *int \$node_id* start of the subtree
- *array \$tree* a pointer to the tree (pointer to save stack space)
- **&\$tree**

recursively determine whether all children of a node are hidden

This walks the subtree starting at \$node_id and returns TRUE if all children of this node are hidden or FALSE if at least 1 is not hidden. As a side-effect, a section/node is made hidden if all children are hidden, i.e. the results are recorded in the nodes in the \$tree.

This recursive routine can be called at most MAXIMUM_ITERATIONS times, preventing endless loops. When this limit is reached, a message is logged (but only once).

Note that the results of subtrees are OR'ed with the is_hidden property of the current node.

This means that IF a subtree has all nodes hidden, the node will be made hidden too. However, if a subtree has at least one non-hidden node, the node will not be forced to be unhidden: if it was already hidden it stays that way.

Class AreaManager

[line 62]

Methods to access properties of an area

This class is used to manage areas. The following functions are supplied

- add a new area (requires PERMISSION_SITE_ADD_AREA)
- set default area (requires PERMISSION_AREA_EDIT_AREA)
- delete existing area (requires PERMISSION_SITE_DROP_AREA)
- edit area properties (requires PERMISSION_AREA_EDIT_AREA)
- view list of areas (requires permissions for add, edit, delete or ACL_ROLE_INTRANET_ACCESS)

The default action is to show a list of existing areas for which the user has some form of permission. This could be either one of the permissions mentioned above or the permission to view the area (intranet).

- **Package** wascore
- **TODO** we need to take care of spurious spaces in inputs (or do we?)

AreaManager::\$areas

array = array() [line 67]

- **Var** list of cached area records keyed with area_id

AreaManager::\$output

object|null = NULL [line 64]

- **Var** collects the html output

AreaManager::\$show_parent_menu

bool = FALSE [line 70]

- **Var** if TRUE the calling routing is allowed to use the menu area (e.g. show config mgr menu)

Constructor *void* function AreaManager::AreaManager(&\$output) [line 79]

Function Parameters:

- *object* **&\$output** collects the html output

construct an AreaManager object

This initialises the AreaManager and also dispatches the chore to do.

- **Uses** \$CFG

void function AreaManager::area_add() [line 287]

present a dialog where the user can enter minimal properties for a new area

this displays a dialog where the user can enter the minimal necessary properties of a new area. These properties are:

- name
- public or private
- the theme to use

Other properties will be set to default values and can be edited later on, by editing the area.

The new area is saved via performing the 'chore' AREAMANAGER_CHORE_SAVE_NEW.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER

void function AreaManager::area_delete() [line 407]

delete an area from this site after confirmation

this either presents a confirmation dialog to the user OR deletes an area. First the user's permissions are checked and also there should be no nodes left in the area before anything is done. Only allowing deletion of an empty area is safety measure: we don't want to accidentally delete many many nodes from an area in one go (see also task_node_delete()). Also, we don't want to introduce orphaned node records (by deleting the area record without deleting nodes).

Note that this routine could have been split into two routines, with the first one displaying the confirmation dialog and the second one 'saving the changes'. However, I think it is counter-intuitive to perform a deletion of data under the name of 'saving'. So, I decided to use the same routine for both displaying the dialog and acting on the dialog.

- **TODO** since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?
- **TODO** should we also require the user to delete any files associated with the area before we even consider deleting it? Or is it OK to leave the files and still delete the area. We do require that nodes are removed from the area, but that is mainly because of maintaining referential integrity. Mmmmm... Maybe that applies to the files as well, especially in a private area. Food for thought.
- **Uses** \$USER

void function AreaManager::area_edit() [line 661]

show the basic properties edit dialog and the edit menu

Note that this dialog does NOT show every area property: the path to the datafiles is missing. It feels too complicated to allow the user to actually change the path because in that case we need to move all existing files to the new location, etc. etc. I did consider to allow the GURU to perform that task (ie editing the path), but eventually decided against it: it is simply not worth it. However, if you know the way to the database and manually edit the path field you can do so, but in that case you're on your own... As a result we don't even show the path to the data directory (it is commented out in the get_dialogdef() routine).

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER

void function AreaManager::area_edittheme() [line 509]

show the theme/area configuration dialog and the edit menu

this displays the list of configurable properties of the theme currently associated with this area in a dialog so that the user can modify the values. Since the area-theme configuration is a more or less 'standard' list of properties, we can use the generic configuration manipulator contained in the ConfigAssistant class.

- **Uses** [ConfigAssistant](#)
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function AreaManager::area_overview() [line 197]

display list of areas with edit/delete icons etc. and option to add an area

this constructs the heart of the area manager: an optional link to add an area followed by a list of links for all areas to which the user has access. From here the user can set the default area, attempt to delete an area and edit the basic and advanced properties of an area. All actions that manipulate an area return here eventually.

Note that the calling routine (the configuration manager) is allowed to display a menu because we set the parameter `show_parent_menu` to `TRUE` here.

The constructed list looks something like this:

```

Add an area
[H] [D] [E] (public) Exemplum Primary School (1, 10)
[ ] [D] [E] (private) Exemplum Intranet (2, 20)
[ ] [D] [E] (public) Exemplum Inactive (3, 30) (inactive)
...

```

The clickable icons [H] and [] manipulate the default area The clickable icons [D] lead to a Delete area confirmation dialog The clickable icons [E] lead to the Edit area (theme parameters) The clickable titles lead to the

Edit area (basic parameters) The clickable link 'Add an area' leads to the add new area dialog.

The area titles are dimmed (grayed-out) if the user is able to see these areas (because they're public or the user has at most intranet access for that area). Private areas for which the user has no access at all don't show up in the list. If the user has Edit-permissions, the area title is not dimmed and the area can be edited.

- **TODO** should we add a paging function to the list of areas? Currently all areas are shown in a single list...
- **TODO** should we make two categories: 'public' and 'private' in the list of areas? Maybe handy when there are many manu areas, but it would be inconsistent with the page manager menu which simply lists the areas in the sort order. Easy way out: the user is perfectly capable to set the sort order in such a way that the sort order already groups the public and private areas. Oh well....
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG
- **Uses** \$USER

void function AreaManager::area_resettheme() [line 575]

reset the theme configuration to the factory defaults

this is a two-step process: we either show a confirmation dialog or we actually overwrite the existing theme configuration with the default values.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$DB
- **Uses** \$CFG

void function AreaManager::area_save() [line 705]

validate and save modified data to database

this saves data from both the edit and the edit theme dialog if data validates. If the data does NOT validate, the edit screen is displayed again otherwise the area overview is displayed again.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function AreaManager::area_savenew() [line 872]

save the newly added area to the database

This saves the essential information of a new area to the database, using sensible defaults for the other fields. Also, a data directory is created and the relative path is stored in the new area record.

If something goes wrong, the user can redo the dialog, otherwise we return to the area overview.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function AreaManager::area_setdefault() [line 328]

make the selected area the default for the site

this sets a default area. First we check permissions and if the user

- is allowed to set the default bit on the target area, AND
- is allowed to reset the default bit on the current default area

We actually

- reset the default bit from the current default (if there is one), AND
- set the default bit for the selected node.

Note: if the user sets the default node on the current default node, the default is reset and subsequently set again (two trips to the database), This also updates the mtime of the record.

- **TODO** should we send alerts? If so, can we use the routine to queue messages from

pagemanager? A reason not to send alerts: the alerts will be sent as soon as a page is added to the new area, so why bother?

- **TODO** should we acknowledge the changed default to the user or is it enough to see the icon 'move'?
- **Uses \$USER**

array function AreaManager::a_param(\$chore, [\$area_id = NULL]) [line 1490]

Function Parameters:

- *string* **\$chore** the next chore that could be done
- *int|null* **\$area_id** the area of interest or NULL if none

shorthand for the anchor parameters that lead to the area manager

int function AreaManager::count_existing_theme_properties(\$area_id, \$theme_id) [line 1593]

Function Parameters:

- *int* **\$area_id**
- *int* **\$theme_id**

determine the number of existing properties for a theme in an area

array function AreaManager::get_dialogdef_add_area() [line 1015]

construct the add area dialog

this constructs an add area dialog definition with the bare minimal fields.

array function AreaManager::get_dialogdef_edit_area(\$area_id) [line 1078]

Function Parameters:

- *int* **\$area_id** indicates for which area

construct the edit area basic properties dialog

Note that this dialog makes the private/public flag readonly; this field is only displayed. Also note that the datadirectory path is shown readonlye too. It is simply too much hassle to

allow the user to change this path because that would imply that the existing files should move along. We'll keep it simple. However, it must be possible to look up the name of the data dir, so therefore we do display it.

void function AreaManager::get_dialog_data(&\$dialogdef, \$record) [*line 1164*]

Function Parameters:

- *array* **&\$dialogdef** contains dialog definition that requires the data
- *array* **\$record** conveniently holds a copy of the area record

fill the dialog with current area data from the database

Note that `area_path` is no longer a part of the `dialogdef` (see also [get_dialogdef_edit_area\(\)](#)) but if it were, the data would still be fetched.

string function AreaManager::get_icon_delete(\$area_id) [*line 1425*]

Function Parameters:

- *int* **\$area_id** the area to delete

construct a clickable icon to delete this area

- **TODO** should we check to see if the area is empty before showing delete icon? Or is it soon enough to refuse deletion when the user already clicked the icon? I'd say the latter. For now...
- **Uses** `$WAS_SCRIPT_NAME`
- **Uses** `$USER`
- **Uses** `$CFG`

string function AreaManager::get_icon_edit(\$area_id) [*line 1458*]

Function Parameters:

- *int* **\$area_id** the area to edit

construct a clickable icon to edit theme properties of this area (edit advanced)

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

string function AreaManager::get_icon_home(\$area_id, &\$areas) [*line 1362*]

Function Parameters:

- *int* **\$area_id**
- *array* **&\$areas** records with area information of all areas

construct a clickable icon to set the default area

the 'default' icon is displayed for the default area, the 'non-default' icon for all others. The user is allowed to make the area the default area if the user has edit permissions for both the old and the new default area.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

array function AreaManager::get_options_themes() [*line 1208*]

fetch a list of themes available for an area

this retrieves a list of themes that can be used as a list of options in a listbox or radiobuttons. Only the active themes are considered. The names of the themes that are displayed in the list are translated (retrieved from the themes language files). The list is ordered by that translated theme name.

array/bool function AreaManager::get_theme_records([*\$forced* = FALSE]) [*line 1249*]

Function Parameters:

- *bool* ***\$forced*** if TRUE forces reread from database (resets the cache)

retrieve a list of all available theme records

this returns a list of active theme-records or FALSE if none are available The list is cached via a static variable so we don't have to go to the database more than once for this. Note that the returned array is keyed with theme_id.

TRUE function AreaManager::reset_theme_defaults(*\$area_id*, *\$theme_id*) [*line 1546*]

Function Parameters:

- *int* ***\$area_id***
- *int* ***\$theme_id***

reset the theme properties of an area to the default values

this deletes any existing properties for the combination of *\$area_id* and *\$theme_id* from the properties table. After that, a copy of the defaults of the theme *\$theme_id* is inserted in the areas-themes properties table. _Eventually_ this may exhaust the available primary keys in the area-theme-properties table. Oh well: with a handful of properties each time you need a lot of resets...

This routine returns TRUE on success or FALSE on error. In the latter case either we were not able to delete old values OR we were not able to insert a copy of the default properties.

void function AreaManager::show_dialog_confirm_delete(*\$area_id*, &*\$areas*) [*line 1272*]

Function Parameters:

- *int* ***\$area_id***
- *array* &***\$areas*** records with area information of all areas

show the name of an area and ask the user for a confirmation of deletion

this displays a confirmation question for deletion of an area. If the user presses Delete button, the area will be deleted, if the user presses Cancel then nothing is deleted.

void function AreaManager::show_edit_menu(\$area_id, [\$current_chore = ""]) [line 1309]

Function Parameters:

- *int* **\$area_id** the area currently being edited
- *string* **\$current_chore** the currently selected edit screen (used to emphasize the option in the menu)

display the edit menu via \$this->output

This displays a clickable menu on in the menu area.

bool function AreaManager::show_parent_menu() [line 139]

allow the caller to use the menu area (or not)

this routine tells the caller if it is OK to use the menu area (TRUE returned) or not (FALSE returned).

int function AreaManager::sort_order_new_area([\$at_begin = FALSE]) [line 1513]

Function Parameters:

- *bool* **\$at_begin** if TRUE the new area is placed before all others, otherwise it is added at the end

determine the value for the sort order of a new area

this calculates a new sort order value based on the existing minimum or maximum values of existing areas (if any).

Note The default sort order for areas differs from that of pages and sections: we assume that the person managing areas knows where to find the newly added area (at the bottom) whereas for a page/section maintainer it is probably more convenient to have a new page/section added at the top of the (perhaps very long) list.

Class ConfigAssistant

[line 259]

class for editing standard configuration tables

Overview

A configuration table works like this: every parameter (property, configuration item) is stored in a record in the configuration table. The core of a configuration table consists of these fields:

- name varchar(240): this is the name of the configuration parameter
- type varchar(2): parameter type: b=bool, c=checkbox, d=date, dt=date/time, f=float(double), i=int, l=list, r=radio, s=string, t=time (see below for more information)
- value text: string representation of parameter value OR a comma-delimited list of values in case of a checkbox
- extra text: a semicolon-delimited list of name=value pairs with additional dialog/validation information, e.g. maxlength=80 or options=true,false,filenotfound (see below for more information)
- sort_order integer: this determines the order in which parameters are presented when editing the configuration
- description text: an optional short explanation of the purpose of this parameter (in English, for internal use only)

There can be additional fields, e.g. links to parent tables in a 1-on-N relation, e.g. themes and themes_properties via theme_id. Also, the configuration table can have a separate primary key to uniquely identify a record but this is not necessary. In the config table the primary key is the name of the parameter.

Parameter types

Here is an overview of the various parameter types. - b=bool:

This type is used to store yes/no type of parameters. The parameter is considered 'TRUE' whenever the integer value of the value field is non-zero. If the integer value of the value field is zero, the parameter is considered to be 'FALSE'. Note that the NULL value also yields a zero integer value and hence 'FALSE'.

- c=checkbox:

This type is an array of boolean parameters. The value of a parameter of this type is stored as a comma-delimited list of values which are to be considered 'TRUE'. If none of the elements of this array are 'TRUE', the comma-delimited list is empty. The list of possible values MUST be specified in the 'extra' field in the 'options=' item. (see below for more information about the 'extra' field).

- d=date:

This type is used to store (valid) dates, in the standard format 'yyyy-mm-dd'. (Validated in [valid_datetime\(\)](#), values from '0000-01-01' - '9999-12-31').

- dt=date/time:

This type is used to store (valid) date/time combinations, in the standard

format 'yyyy-mm-dd hh:mm:ss'. (Validated in [valid_datetime\(\)](#), values from '0000-01-01 00:00:00' - '9999-12-31 23:59:59').

- f=float(double):

This type is used to store real (floating point) numbers with double precision.

- i=int:

This type is used to store integer numbers.

- l=list:

This type is used to store a single value from a list of available options (a 'picklist'). The current value is stored in the value field, and a list of possible values MUST be specified in the 'extra' field in the 'options=' item. (see below for more information about the 'extra' field).

- r=radio:

This type is also used to store a single value from a list of available options, much like the list-type (a 'picklist'). The difference is the representation in a dialog: a list parameter uses only a single line, whereas a group of radio buttons usually uses as many lines as there are available options. The current value is stored in the value field, and a list of possible values MUST be specified in the 'extra' field in the 'options=' item. (see below for more information about the 'extra' field).

- s=string:

This type is used to store generic text information. The maximum length of the string is the maximum length of the text field in the database (in MySQL this is 65535 bytes).

- t=time:

This type is used to store a (valid) time, in the format 'hh:mm:ss'. (Validated in [valid_datetime\(\)](#), values from '00:00:00' - '23:59:59').

The Extra field

This field can contain additional information about the parameter, either for validation or for display purposes. The contents of this field is a list of semicolon-delimited name=value-pairs. The following items are recognised (see also [dialoglib.php](#) - rows=<int>

This is the number of rows to display in a dialog. It can apply to string-type variables and yield a textarea-tag (as opposed to an input-tag of type 'text'). It can also apply to a set of radio buttons in a very specific way: if the number of rows is 1, the radio buttons are displayed on a single line in the dialog.

- columns=<int>

This is number of columns to use for input (but not the necessary the maximum length of the input). If this item is omitted, default values apply, e.g. 30 for date, time, datetime; 20 for float (double) and 10 for integer, etc.

- minlength=<int>
This is the minimum number of characters that must be input. When this item is set to 1, an empty string is not allowed.
- maxlength=<int>
This is the maximum number of characters that can be input.
- minvalue=<mixed>
This is the minimum value for the field. The type of the minimum value is the same as the type of the variable itself. It applies to integers, floats, dates, datetimes and times.
- maxvalue=<mixed>
This is the maximum value for the field. The type of the maximum value is the same as the type of the variable itself. It applies to integers, floats, dates, datetimes and times.
- decimals=<int>
This is the number of decimals that should be displayed in dialogs. It applies to floats.
- options=option1,option2,option3,...
This is a comma-delimited list of valid values for a list, radio or checklist parameter. The value of the parameter is one of the options in this list (for parameter types list and radio) OR a comma-delimited list of zero or more items (for checklists).
- viewonly=<int>
If the integer value of this item is non-zero, the user is not allowed to edit the value of the parameter. However, it is supposed to be displayed in the dialog nevertheless.

Generating dialogs for editing

The ConfigAssistant is clever enough to read and write the configuration parameters from the specified table, using a where-clause when necessary. Also, the ConfigAssistant automatically constructs translations of screen prompts in a very specific way when constructing dialogs.

The translation keys are constructed as follows. - simple string-like parameters (string, date, time, etc.)

```
name = <prefix><name>
label = <prefix><name>_label
title = <prefix><name>_title
```

- bool parameter
name = <prefix><name>

```
label = <prefix><name>_label  
title = <prefix><name>_title  
option = <prefix><name>_option
```

- list or radio parameter

```
name  = <prefix><name>  
label = <prefix><name>_label,  
title = <prefix><name>_title  
options:  
label = <prefix><name>_<option1>_label  
title = <prefix><name>_<option1>_title  
...  
label = <prefix><name>_<optionN>_label  
title = <prefix><name>_<optionN>_title
```

- checklist parameter

```
name  = <prefix><name>  
label = <prefix><name>_label  
title = <prefix><name>_title  
options:  
title = <prefix><name>_<option1>_title  
option = <prefix><name>_<option1>_option  
...  
title = <prefix><name>_<optionN>_title  
option = <prefix><name>_<optionN>_option
```

The string <prefix> can be used to avoid name clashes in the 'admin' (or other) language file. The translations are then looked up in the specified language domain (default: admin).

Examples

Example 1: sending a dialog to the user for editing all the parameters in the the main config table:

```
$table = 'config';  
$keyfield = 'name';  
$assistant = new ConfigAssistant($table,$keyfield);  
$href = 'index.php?job=(...)&task=editconfig';  
$assistant->show_dialog($output,$href);
```

Example 2: saving the modified data to the table

```
$table = 'config';  
$keyfield = 'name';  
$assistant = new ConfigAssistant($table,$keyfield);  
if (!$assistant->save_data($output)) {  
    echo "FAILED";  
} else {
```

```
    echo "SUCCESS saving configuration";  
}
```

Sounds easy to use, doesn't it?

- **Package** wascore
- **TODO** implement checklist
- **Usedby** [AreaManager::area_edittheme\(\)](#)

ConfigAssistant::\$dialogdef

array = NULL [line 282]

- **Var** \$dialogdef an array with a dialog ready to use for [dialog_quickform\(\)](#)

ConfigAssistant::\$dialogdef_hidden

array = [line 285]

- **Var** \$dialogdef_hidden array with additional fields that should be included in the dialog

ConfigAssistant::\$fields

array = array('name','type','value','extra') [line 264]

- **Var** \$fields the list of essential fields to retrieve from the the table

ConfigAssistant::\$keyfield

string = [line 270]

- **Var** \$keyfield a string indicating the keyfield to uniquely identify the configuration parameter

ConfigAssistant::\$language_domain

\$string = [line 279]

- **Var** \$domain the language domain where to look for translations (default: 'admin')

ConfigAssistant::\$prefix

\$string = [line 276]

- **Var** \$prefix is prepended for every translation/language key and the dialog item name

ConfigAssistant::\$records

array = NULL [line 273]

- **Var** \$records the cached list of configuration values straight from the database

ConfigAssistant::\$table

string = [line 261]

- **Var** \$table the table that contains the configuration

ConfigAssistant::\$where

mixed = [line 267]

- **Var** *\$where* a string with a whereclause (without 'WHERE') or an array with conditions

Constructor *void* function ConfigAssistant::ConfigAssistant(\$table, \$keyfield, [\$prefix = ''], [\$domain = ''], [\$where = ''], [\$dialogdef_hidden = '']) [line 300]

Function Parameters:

- *string* **\$table** the table where the configuration parameters are stored
- *string* **\$keyfield** the field that uniquely identifies the configuration parameters
- *string* **\$prefix** is prepended for every translation/language key and the also dialog item name
- *string* **\$domain** the language domain where to look for translations (default: 'admin')
- *mixed* **\$where** a whereclause (without 'WHERE') or an array with contions
- *array* **\$dialogdef_hidden** additional fields for inclusion in dialog definition

constructor for the configuration assistant

This stores the parameters, sets defaults when applicable and subsequently reads selected config parameters into the *\$this->records* for future reference.

array function ConfigAssistant::get_dialogdef() [line 391]

construct an array with the dialog information

- **TODO** implement checklist

array function ConfigAssistant::get_extra(\$type, \$extras) [line 500]

Function Parameters:

- *string* **\$type** variable type (necessary for calculating minvalue/maxvalue)
- *string* **\$extras** semicolon-delimited list of name=value pairs

construct an array based on name=value pairs in an 'extra' field

This constructs an array based on the name=value pairs in the extras string. Most recognised parameters yield an integer value. Exceptions are: minvalue and maxvalue: these yield a variable of the same type as the config parameter itself options yields an array with all options from the comma delimited list Unknown name=value pairs are logged with LOG_DEBUG.

void function ConfigAssistant::get_options_from_extra(\$extra, \$name) [*line 542*]

Function Parameters:

- **\$extra**
- **\$name**

void function ConfigAssistant::save_data(&\$output) [*line 337*]

Function Parameters:

- *object* **&\$output** the object that collects the output

save the modified configuration parameters to the database

- Uses [dialog_validate\(\)](#)

void function ConfigAssistant::show_dialog(&\$output, \$href) [*line 324*]

Function Parameters:

- *object* **&\$output** the object that collects the output
- *string* **\$href** the target for the form that will be created

add a complete dialog to the content area of the output

- Uses [dialog_quickform\(\)](#)

Class DatabaseMysql

[line 58]

MySQL database

This implements access to the MySQL database.

- **Package** wascore

DatabaseMysql::\$db_link

resource = [line 78]

- **Var** database link identifier

DatabaseMysql::\$db_name

string = [line 72]

- **Var** database to use, e.g. 'was'

DatabaseMysql::\$db_password

string = [line 69]

- **Var** part of credentials for database access

DatabaseMysql::\$db_server

string = [line 63]

- **Var** database server, e.g. 'localhost' or 'db.example.com:3306'

DatabaseMysql::\$db_type

string = [line 60]

- **Var** database type, e.g. 'mysql'

DatabaseMysql::\$db_username

string = [line 66]

- **Var** part of credentials for database access

DatabaseMysql::\$debug

bool = [line 90]

- **Var** if true switch debugging on

DatabaseMysql::\$errno

integer = [line 84]

- **Var** the error number generated by the latest mysql command

DatabaseMysql::\$error

string = [line 87]

- **Var** the error message generated by the latest mysql command

DatabaseMysql::\$prefix

string = [line 75]

- **Var** table name prefix, e.g. 'was_'

DatabaseMysql::\$query_counter

integer = [line 81]

- **Var** the number of queries executed so far

Constructor *void* function DatabaseMysql::DatabaseMysql(\$prefix, [\$debug = FALSE]) *[line 100]*

Function Parameters:

- *string* **\$prefix** table name prefix, e.g. 'was_'
- *bool* **\$debug** if TRUE extra information is displayed (handy for debugging the code)

initialise query counter and other variables, store the table prefix

bool function DatabaseMysql::close() [*line 143*]
close the connection to the database

bool/string function DatabaseMysql::column_definition(\$fielddef) [*line 526*]
Function Parameters:

- array **\$fielddef** an array that describes a field in a generic way

convert a fielddef array to a MySQL specific column definition

this creates a MySQL specific column definition based on a generic field description. The following keywords are recognised:

- name: the name of the field (required)
- type: the generic type of the field (see table below for supported types) (required)
- length: the size of a numeric field or the length of a textual field
- decimals: the number of decimals in real numbers (implies length)
- unsigned: used for numeric fields that don't need negative values (see note 5 below)
- notnull: if true, the field is not allowed/able to contain the NULL value
- default: the default value for the field
- enum_values: an array with allowable values for the enum field
- comment: a string that can be used to document the field definition

The generic field types are mapped to actual MySQL data types via a lookup table. Most types have a number of boolean flags which indicate how the field definition must be interpreted. Entries with an empty type are considered special cases.

- len = 1: look for a length parameter and use it if it is defined.
- dec = 1: look for a decimals parameter and use it if it is defined. Implies len.
- unsigned = 1: look for an unsigned parameter and use it if it is defined and true (see note 5 below).
- default = 1: look for a default parameter, and use it if it is defined.
- quote = 1: if a default is allowed AND defined, use single quotes + escaped string.

Note 1: a 1 in the table means that the field type `_allows_` the corresponding parameter, and 0 means that this parameter is NOT allowed. It doesn't say anything about parameters being `_required_` (e.g. a varchar must have a length). It is the responsibility of the author of the field definition to provide all necessary parameters.

Note 2: `enum_values` are not used at this time; an enum-field simply maps to a varchar field and that's it. Adding the `enum_values` to a data definition does help to document the purpose of the field. The reason for not (yet) implementing enums in full is the issues associated with the translations in the UI. Furthermore, using native enums in MySQL is a royal PITA. For now the application should know what it is doing when using and updating enums in a table.

Note 3: at this time it is not possible to set the default value of a text-type field to the string consisting of the letters N, U, L and L: that caseinsensitive string is always interpreted as the NULL-value in the database, ie. it yields "DEFAULT NULL" and not "DEFAULT 'NULL'" or "DEFAULT 'Null'" or "DEFAULT

'null'".

Note 4: even though comments are not yet stored in the MySQL database, the corresponding COMMENT-clauses are generated, if only for documentation/debugging purposes. These clauses are correctly parsed by MySQL but they are subsequently discarded.

Note 5: As of version 2009051401 (0.0.5) the parameter 'unsigned' is deprecated. This is because even though MySQL implements an unsigned attribute for numeric fields, other RDBMSs might not. Therefore we stay away from this MySQL-specific construct as much as possible. Also, the translation from the 'serial' fieldtype no longer adds the unsigned attribute to the actual MySQL definition.

- **TODO** should we allow both int and integer?
- **TODO** 'enum' type equivalent with varchar, enum_values[] array is not used at all, only as a form of documentation

concatenation function DatabaseMysql::concat(\$string1, \$string2) [line 231]

Function Parameters:

- *string* **\$string1** contains a quoted/escaped string, a fieldname or other expression
- *string* **\$string2** contains a quoted/escaped string, a fieldname or other expression

helper function for string concatenation in sql statements

From <http://troels.arvin.dk/db/rdbms/#functions-concat>:

SQL Standard: Core feature ID E021-07: Concatenating two strings is done with the || operator:

string1 || string2

If at least one operand is NULL, then the result is NULL.

MySQL: Badly breaks the standard by redefining || to mean OR. Offers instead a function, CONCAT(string, string), which accepts two or more arguments.

In order to try and stay as database-independent as possible without losing this concatenation feature, we have to resort to a database-specific function. Aaargggghh!

Typical use of this function:

```
...
$sql = 'UPDATE {'$DB->prefix}table '.
      'SET message = ' . $DB->concat('message', "'". $DB->escape("addition to
message\n")."'").' '.
      'WHERE table_id = ' . $some_id;
$DB->exec($sql);
...
```

and that's exactly the kind of hairy code I'd like to stay away from, because of the necessary delicate balancing of quotes and the amount of thought it requires to get a query right. Obviously it is much easier (and less quote-error prone and almost elegant) to do something like this:

```
$record = db_select('tablename', 'message', array('table_id' => $some_id));
$value = $record['message'] . "addition to message\n";
db_update('tablename', array('message' => $value), array('table_id' => $some_id));
```

Note that 'Standard-SQL' would yield almost as much trouble with concatenation, even if it were possible to use in MySQL, e.g.:

```
...
$sql = 'UPDATE {'$DB->prefix}table '.
      "SET message = message || '". $DB->escape("addition to message\n")."'
      'WHERE table_id = ' . $some_id;
$DB->exec($sql);
...
```

No matter what: this is ugly. The reason I still want to use this kind of code is that (much) more expensive to use series of SELECT / UPDATE statements with concatenation in PHP, not to mention the fact that using two separate SQL-statements introduces race conditions, so there. Alas this is database specific.

Note that some of the quote-hell can be dealt with via [db_escape_and_quote\(\)](#):

```
$sql = 'UPDATE {'$DB->prefix}table '.
      'SET message = ' . $DB->concat('message', db_escape_and_quote("addition to
message\n")).' '.
      'WHERE table_id = ' . $some_id;
$DB->exec($sql);
```

- **TODO** perhaps extend this function to accept more than 2 strings?

bool function DatabaseMysql::connect(\$db_server, \$db_username, \$db_password, \$db_name) [*line 119*]

Function Parameters:

- *string* **\$db_server** database server, e.g. 'localhost' or 'db.example.com:3306'
- *string* **\$db_username** part of credentials
- *string* **\$db_password** part of credentials
- *string* **\$db_name** database to use, e.g. 'was'

connect to the database server and open the database

- **TODO** weigh pros and cons of persistent database connections, perhaps add as config option?

int/bool function DatabaseMysql::create_table(\$tabledef) [*line 396*]

Function Parameters:

- *array* **\$tabledef** a generic table definition (not database-specific)

create a table via a generic (non-MySQL-specific) table definition

this executes a MySQL-specific CREATE TABLE statement based on a generic table definition. The actual work is done in create_table_sql(), which makes it possible to see the result of converting a generic definition to an actual table; very handy while debugging.

- **TODO** document correct link for documentation of generic table definition 'tabledefs.php'

string/bool function DatabaseMysql::create_table_sql(\$tabledef) [*line 415*]

Function Parameters:

- *array* **\$tabledef** a generic table definition (not database-specific)

create the MySQL-specific SQL statement to create a table via a generic table definition

this creates a MySQL-specific CREATE TABLE statement from a generic table definition. See [tabledefs.php](#) for more information about the format of this generic table definition.

- **TODO** document correct link for documentation of generic table definition 'tabledefs.php'
- **TODO** find a way to deal with the enum values: where do we keep them? Or do we keep them at all?

int/bool function DatabaseMysql::drop_table(\$tablename) [line 360]

Function Parameters:

- *string* **\$tablename** name of the table to drop (prefix will be added automatically)

unconditionally drop the specified table

bool function DatabaseMysql::dump(&\$data, [\$drop = TRUE], [\$tables = ""]) [line 653]

Function Parameters:

- *string* **&\$data** receives the dump of the tables
- *bool* **\$drop** if TRUE add code to drop the table before the data definition
- *mixed* **\$tables** array with names of tables to dump, empty (string, array) means all our tables

make a text dump of our tables in the database suitable for backup purposes

This creates a text dump of the selected tables in parameter \$data. If \$tables is empty we dump all the tables in the database that start with 'our' prefix (there could be other websites using the same table with another prefix, we won't dump those). If \$tables is an array, it is assumed to be an array with table names without our prefix. In this case we will prepend the prefix. If parameter \$drop is TRUE, we add code to drop the table from the database (if it exists) before we recreate it.

If there were no errors, we return TRUE (and \$data contains the dump). If errors were encountered, we return FALSE and \$this->errno and \$this->error can tell the caller what happened. If there were errors, \$data is undefined.

Strategy is as follows. First we make a valid list of tables to dump, either by asking the database for a list of tables LIKE "{\$prefix}%" or by manipulating and validating the array \$tables that was provided by the caller.

Subsequently we let the database generate a CREATE TABLE statement and then we step through the data (if any) and add it to \$data.

Note MySQL is quite liberal in what it accepts as field values. However, I try to generate INSERT INTO-statements as clean as possible by NOT quoting numeric values. HTH. It still is a MySQL-specific dump, though. You cannot simply use the result 'as-is' to migrate to another database.

string/bool function DatabaseMysql::escape(\$unesaped_string) [line 156]

Function Parameters:

- *string* **\$unesaped_string** the string to escape

escape special characters in string

int|bool function DatabaseMysql::exec(\$sql) [*line 249*]

Function Parameters:

- *string* **\$sql** valid SQL statement

execute an action query and return the number of affected rows

this method should be used to execute SQL-statements that do NOT return a result set, use query() for that. This method works well for INSERTs, DELETEs and UPDATEs. If there is an error this method returns FALSE. Otherwise it returns the number of affected lines. Note that there is a difference between 0 affected lines and FALSE.

int|bool function DatabaseMysql::last_insert_id([\$table_name = "], [\$field_name = "]) [*line 297*]

Function Parameters:

- *string* **\$table_name** (optional) tablename (unused in MySQL)
- *string* **\$field_name** (optional) fieldname (unused in MySQL)

retrieve the most recent automatically inserted id ('auto_increment')

this method returns the id that was automatically generated in the previous INSERT-query or 0 if the previous query was not an INSERT query.

Note: as per the MySQL manual this function returns an int and not a bigint. If the auto_increment field is a bigint, this method returns an incorrect result. There is a work-around (e.g. SELECT LAST_INSERT_ID()) but this is not the way it works in Website@School; all id's are simple int's and not bigint's, so there is no need to generate yet another query after every insert.

Note that this method can be called with a table name and a field name. This is a hook for future expansion; this MySQL-driver does not actually use it. However, it is handy to always call this method with table and field name to make adding a new database driver easier.

object|bool function DatabaseMysql::query(\$sql, [\$limit = "], [\$offset = "]) [*line 328*]

Function Parameters:

- *int* **\$offset** optional number of records to skip; = 0 means start with the first
- *string* **\$sql** valid SQL statement
- *int* **\$limit** optional limitation of the number of records returned

execute a select query and return a result set

this method should be used to execute SQL-statements that do return a result set: SELECT, SHOW, EXPLAIN and DESCRIBE. Use exec() for action queries . If there is an error this method returns FALSE. Otherwise it returns a result set in the form of a DatabaseMysqlResult object.

if parameters \$limit and \$offset are set the result set contains at most \$limit rows, starting at offset \$offset. it is OK to specify just \$limit; \$offset is taken into account only when \$limit is specified too. Note that different databases have a different syntax for limiting the number of returned records. MySQL supports both 'LIMIT limit OFFSET offset' (which we use here) and 'LIMIT offset,limit'.

the LIMIT-clause is blindly appended to the SQL-statement; it is up to the caller to decide wheter specifying a limit and an offset make sense or not.

bool function DatabaseMysql::table_exists(\$tablename) [*line 371*]

Function Parameters:

- *string* **\$tablename** name of the table to check (prefix will be added automatically)

see if the named table exists

Class DatabaseMysqlResult

[*line 772*]

MySQL database result

This implements access to database result sets

- **Package** wascore

DatabaseMysqlResult::\$errno

integer = [line 777]

- **Var** the error number generated by the latest mysql command

DatabaseMysqlResult::\$error

string = [line 780]

- **Var** the error message generated by the latest mysql command

DatabaseMysqlResult::\$num_rows

integer = [line 783]

- **Var** the number of rows in the result set

DatabaseMysqlResult::\$result

resource = [line 774]

- **Var** the resource associated with the result set

Constructor *void* function DatabaseMysqlResult::DatabaseMysqlResult(\$result) *[line 789]*

Function Parameters:

- *resource* **\$result** the resource associated with the result set

constructor

bool function DatabaseMysqlResult::close() [*line 801*]

free the memory associated with the result set

array function DatabaseMysqlResult::fetch_all() [*line 831*]

fetch all rows as a 0-based array of 0-based enumerated arrays

array function DatabaseMysqlResult::fetch_all_assoc([\$keyfield = ""]) [*line 851*]

Function Parameters:

- *string* **\$keyfield** field to use as the key in the returned array or empty for 0-based numeric array key

fetch all rows as an array (0-based or keyed) of associative arrays

This returns an array of assoc arrays (one per record). If \$key is not empty, we use the contents of the field as the array key, otherwise we simply use a 0-based numeric key. By specifying a single unique field (e.g. the primary key) all records in the array can be accessed via their unique value.

array/bool function DatabaseMysqlResult::fetch_row() [*line 813*]

fetch the next result row as a 0-based enumerated array

array/bool function DatabaseMysqlResult::fetch_row_assoc() [*line 822*]

fetch the next result row as a associative array

Class Email

[*line 46*]

Email implements a simple interface to send mail

This class can be used to send mail from Website@School, e.g. alerts, new passwords,

feedback to the project (with translations), etc.

```
Typical use: require_once('email.class.php');
$mailer = new Email;
$mailer->set_mailto($email,$name);
$mailer->set_subject($subject);
$mailer->set_message($message);
$mailer->add_attachment($data,$name);
$mailer->send();
```

- **Package** wascore

Email::\$attachments

array = array() [line 69]

- **Var** \$attachments array of arrays with attachment properties: body, name, mimetype, charset, encoding

Email::\$charset

string = UTF-8 [line 75]

- **Var** \$charset default character set to use in display names and subject

Email::\$eol

string = \r\n [line 72]

- **Var** \$eol end of line character(s), usually CR + LF

Email::\$headers

array = array() [line 63]

- **Var \$headers** associative array with field names and field values of additional headers

Email::\$mailcc

array = array() [line 57]

- **Var \$mailcc** contains an array of arrays containing addr and name for Cc: (array of addresses)

Email::\$mailfrom

array = array() [line 48]

- **Var \$mailfrom** contains addr and name for From: (single address)

Email::\$mailreplyto

array = array() [line 51]

- **Var \$mailreplyto** contains addr and name for Reply-To: (single address)

Email::\$mailto

array = array() [line 54]

- **Var \$mailto** contains addr and name for To: (single address)

Email::\$max_length

int = 76 [line 81]

- **Var** \$max_length limit for line length

Email::\$message

array = array() [line 66]

- **Var** \$message associative array with message properties: body, mimetype, charset, encoding

Email::\$minimal

bool = FALSE [line 78]

- **Var** \$minimal default value of flag for limiting the literal representation in [rfc2047_qchar\(\)](#)

Email::\$subject

string = [line 60]

- **Var** \$subject contains the message subject

Constructor *void* function Email::Email() *[line 87]*

constructor resets all variables to a known (default) state

void function Email::add_attachment(\$attachment, \$name, [\$mimetype = 'application/octet-stream'], [\$charset = 'UTF-8'], [\$encoding = 'base64']) *[line 223]*

Function Parameters:

- *string* **\$attachment** presumably 8bit data to attach to the message
- *string* **\$name** the suggested filename to use when receiving the attachment
- *string* **\$mimetype** type of the content, usually the generic 'application/octet-stream'
- *string* **\$charset** character set to use (only applicable when \$mimetype indicates 'text')
- *string* **\$encoding** the desired encoding (defaults to base64 because we expect binary data)

add an attachment

This simply adds an attachment with associated properties. Multiple attachments can be added by calling this routine multiple times.

void function Email::add_mailcc(\$addr, [\$name = ""]) [*line 183*]

Function Parameters:

- *string* **\$addr** the address eg. 'acackl@example.com'
- *string* **\$name** the name, eg. 'Amelia Cackle'

add an address and name for the Cc: header

Embedded CRs LFs "<" and ">" are removed from \$addr, and the result is stored, together with \$name. Note that this function can be called multiple times, where each call adds an address to the list.

bool function Email::is_7bit(\$source) [*line 684*]

Function Parameters:

- *string* **\$source** the text to examine

a small utility routine to determine if a string has only 7bit characters

- **Usedby** [Email::rfc2047_qstring\(\)](#)
- **Usedby** [Email::send\(\)](#)

`void function Email::reset_all()` [line 95]

reset all variables to their default values

`string function Email::rfc2047_qchar($c, &$required_len, [$minimal = FALSE])` [line 654]

Function Parameters:

- *int* **\$c** the character to encode
- *int* **&\$required_len** returns the space required for this encoded char (UTF8-aware)
- *bool* **\$minimal** if TRUE, only [0-9A-Za-z] use literal representation, otherwise encoding is more relaxed

encode an 8-bit byte according to Q-encoding in RFC2047

This routine encodes a single integer ASCII code into either

- literal representaion
- generic 8bit representation, ie. "=" followed by 2 (uppercas)e hexdigits
- an underscore character

If \$minimal is FALSE, all printable ASCII characters from 33 "!" to 126 "~" except 61 "=", 63 "?" and 95 "_" use literal representation. Character 32 " " is represented as an underscore (for improved readabilitu/deciphering).

If \$minimal is TRUE, only digits "0" - "9" and letters "A" - "Z" and "a" - "z" use literal representation and character 32 " " uses generic 8bit encoding "=20".

The latter case yields only digits, letters, equal-sign and question mark, which should travel undisturbed through any mail transdfer agent.

There is a special situation when encoding UTF8 where characters can span multiple octets. The length of such a sequence can be determined by the number of most significant 1's in a row in the first octet. If \$c is the first octet of a UTF8-sequence, we tell the caller the total length of the encoded sequence, not just the length of the encoded 1st octet (which would always be 3). This forces the caller to start a new 'encoded-word' with enough room for the complete sequence if necessary, preventing a multi-octet sequence to span two 'encoded-words'. Note that characters in a UTF8-tail yield length 3, even when more UTF8-tail octets follow. That is OK because the first character already 'reserved' the space when the first octet was processed.

Here is a small truth table for sequence lengths (see also RFC3629).

bit pattern	range	len	comments
0xxx.xxxx	0-127	3	ASCII
10xx.xxxx	128-191	3	octet is part of UTF8-tail
110x.xxxx	192-223	6	UTF8-2, beginning of a sequence of 2 octets
1110.xxxx	224-239	9	

UTF8-3, beginning of a sequence of 3 octets 1111.0xxx 240-247 12 UTF8-4, beginning of a sequence of 4 octets 1111.10xx 248-251 3 sequence of 5 characters not defined in RFC3929, settle for length 3 1111.110x 252-253 3 sequence of 6 characters not defined in RFC3929, settle for length 3 1111.111x 254-255 3 no sequence at all, settle for length 3

Note that if \$c is NOT UTF8 but say ISO-5988-1, the worst that can happen is that a perfectly valid single octet character in the range 192-247 would indicate a length of more than the necessary 3, pushing up to 4 characters to the next 'encoded-word'. Oh well, I can live with that.

References: <http://www.ietf.org/rfc/rfc2047.txt>, <http://www.ietf.org/rfc/rfc3629.txt>.

- Used by [Email::rfc2047_qstring\(\)](#)

string function Email::rfc2047_qstring(\$source, &\$remaining, [\$charset = "UTF-8"], [\$minimal = FALSE], [\$max_length = 76], [\$eol = "\r\n"]) [*line 543*]

Function Parameters:

- *string* **\$source** the string to encode (could be 7bit)
- *int* &**\$remaining** the number of bytes remaining on the current output line (not counting any CR+LF)
- *string* **\$charset** indicates character set used in \$source
- *bool* **\$minimal** if TRUE, rfc2047_qchar() limits literal encoding to digits and letters
- *int* **\$max_length** limit on output line length (and indirect of the 'encoded-word' length) to max 76 (75)
- *string* **\$eol** the end of line character(s), default as per RFC5322 (RFC822) is CR chr(13) + LF chr(10)

encode a string according to RFC2047 (Message Header Extensions for Non-ASCII Text)

This routine encodes \$source according to RFC2047 (Message Header Extensions for Non-ASCII Text) using the 'Q'-encoding (somewhat comparable to quoted_printable). However, if the \$source uses only harmless 7bit characters and falls within the limit of \$remaining characters it is returned unchanged and the number of \$remaining characters is updated accordingly.

In all other cases (\$source contains bytes > 127, \$source is longer than \$remaining, etc.) this encodes the string into 'encoded-word's of max 75 chars. These 'encoded-word's look like this: "=?" charset "?" encoding "?" encoded-text "=?" with 'encoding' always equal to "Q" (similar to quoted printable). The actual encoding of characters is done in [rfc2047_qchar\(\)](#). The boolean flag \$minimal can be used to limit the literal representation to only digits and letters, using generic 8bit encoding by setting it to TRUE.

If multiple 'encoded-word's are necessary, they are separated from each other by a folding space, i.e. newline (using \$eol) followed by a normal space (ASCII 32). The end result never ends with such a folding space; the returned value always ends with the "=?" of the last 'encoded-word'.

Note 1: I found it quite hard to read the combination of RFC5322 and RFC2047 because I had some trouble distinguishing the rules for RFC5322-type headers. I finally settled for this simplified set

- From:, To:, Cc: and Reply-To: are all of type 'mailbox' to me (KISS, no 'group's and stuff)

- A 'mailbox' can be either written as ['display-name'] "<" addr-spec ">" OR as
addr-spec "(" ctext ")", which both allow for FWS (folding white space)

- A Subject: field is simply 'unstructured', which also allows for FWS

Furthermore, this routine simply encodes non-ASCII text and therefore makes no assumptions about the contents of \$source; it is the caller's responsibility to make sure that the 'ctext' or 'unstructured' or 'display-name' conforms to RFC5322.

Note 2: I have not implemented fancy and streamlined code to minimise the amount of encoded characters, ie. leaving pure ASCII-words unencoded and encoding only non-ASCII words or words containing "=?" because I could not invest that amount of time.

Maybe in a later version... (famous last words). I did optimise for short and pure and simple ASCII strings because I expect that generated Subject: headers and other headers will be ASCII most of the time. We'll see how that works out.

Note 3: Quirk: if initially there is not enough space for the shortest possible 'encoded-word', we insert a FWS even if \$source has no WSP at that point. Basically it means that we add a character to the result. This may or may not be a problem for the caller. OTOH: the caller should provide enough space in the first place, so there.

References: see <http://www.ietf.org/rfc/rfc2047.txt> and <http://www.ietf.org/rfc/rfc5322.txt>.

- **TODO** maybe optimise this routine to let pure ASCII-words through unencoded (in a later version)
- **Usedby** [Email::rfc5322_address\(\)](#)

- **Usedby** [Email::send\(\)](#)
- **Uses** [Email::rfc2047_qchar\(\)](#)
- **Uses** [Email::is_7bit\(\)](#)

string function `Email::rfc5322_address($addr_spec, &$remaining, [$display_name = "], [$legacy = FALSE], [$charset = "UTF-8"], [$minimal = FALSE], [$max_length = 76], [$eol = "\r\n"])` [line 746]

Function Parameters:

- *string* **\$addr_spec** is an address of the form 'local-part' "@" 'domain' (RFC5322 section 3.4.1)
- *int* **&\$remaining** indicates how much space is left on the current output line
- *string* **\$display_name** is the human readable name associated with \$addr_spec
- *bool* **\$legacy** if TRUE, output is in the legacy format 'addr-spec' "(" 'cctx'")"
- *string* **\$charset** indicates character set used in \$display_name
- *bool* **\$minimal** if TRUE, eventually `rfc2047_qchar()` limits literal encoding to digits and letters
- *int* **\$max_length** limit on output line length (and indirect of the 'encoded-word' length) to max 76 (75)
- *string* **\$eol** the end of line character(s), default as per RFC5322 (RFC822) is CR `chr(13)` + LF `chr(10)`

construct an address field according to RFC5322 (RFC822)

This routine constructs an address according to (simplified) rules in RFC5322 section 3.4. Depending on the \$legacy flag, the parameters are used to construct either an 'angle-addr' `DQUOTE $display_name DQUOTE SPACE "<" $addr_spec ">"` or an address with the display-name "hidden" in a comment `$addr_spec SPACE "(" $display_name ")"`

Basically the \$addr_spec is not modified; it is assumed that this string obeys the rules for 'addr-spec' in RFC5322. Specifically we do not encode this information (with [rfc2047_qstring\(\)](#) or otherwise). However, we DO strip any CR and/or LF characters because these might cause problems lateron (eg. an unwanted extra blank line in the mail headers). Also, we definately do not want to have angle brackets, so we remove those too, just to be sure.

The \$display_name can be modified. Reading RFC5322 yields the following (simplified) rules. 'display-name' => 'phrase' => 1*'word'; 'word' => 'atom' | 'quoted-string'. In other words: it is allowed to use a quoted string, ie.

- a `DQUOTE`, followed by

- a string with printable ASCII characters not being DQUOTE or the quote character backslash, followed by
- a DQUOTE.

Note that FWS (folding white space) is allowed between the two DQUOTEs. This leads an easy way out of to stripping DQUOTEs and backslashes before the 'display-name' is encoded and/or folded.

If the legacy-flag is set, we use the parameter \$display_name to construct a (simplified) comment, ie.

- a "(", followed by
- a string of ctext characters not containing "(", ")" or a backslash, followed by
- a ")".

Here, too, folding whitespace is allowed between the opening "(" and closing ")". This variant leads the easy way out of stripping parentheses and backslashes before the 'display-name' is encoded and/or folded as a comment.

Note: All this cleaning up of CR, LF, DQUOTE, etc. does NOT weed out other CTLs (ASCII 0...ASCII 31).

- **Usedby** [Email::send\(\)](#)
- **Uses** [Email::rfc2047_qstring\(\)](#)

string function [Email::rfc5322_message_id\(\)](#) [*line 818*]

construct a message-id conforming to RFC5322 (RFC2822, RFC822)

This constructs a message id according to specifications in section 3.6.4 in RFC5322 Internet Message Format (October 2008), see <http://www.ietf.org/rfc/rfc5322.txt>.

- **Usedby** [Email::send\(\)](#)

bool function [Email::send\(\)](#) [*line 317*]

send the message using the prepared information (To:, Subject:, the message and attachments etc.)

This actually sends the message (using PHP's mail() command), using all the prepared data & headers etc.

Depending on the contents of the message and the number of added attachments, there are the following possibilities.

1. the message is a single plain 7bit ASCII-text with lines shorter than 76 characters, no attachments
2. the message is a text with either 8bit values OR lines longer than 76 characters but still no attachments
3. there are attachments too
 - Ad 1: no need for MIME in that case; simply use the message as-is, using the default
 - 7bit encoding and US-ASCII charset. Easy
 - Ad 2: need MIME but not multipart: headers to add:
 - MIME-Version: 1.0
 - Content-Type: text/plain; charset="UTF-8"
 - Content-Transfer-Encoding: quoted-printable
 - Ad 3. need MIME Multipart; headers to add:
 - MIME-Version: 1.0
 - Content-Type: multipart/mixed; boundary="\$boundary"and in the body we need to construct the sequence of message and 1 or more attachments, each with their own headers like
 - Content-Type: text/plain; charset="US-ASCII"
 - Content-Transfer-Encoding: 7bitor
 - Content-Type: text/plain; charset="UTF-8"
 - Content-Transfer-Encoding: quoted-printableand for the attachments (could be more than 1):
 - Content-Type: application/octet-stream; name="\$name"
 - Content-Transfer-Encoding: base64
 - Content-Disposition: attachment; filename="\$name"and of course with a \$boundary between the various parts of the message

Note that we listen to the caller most of the time, ie. if the caller specifies an attachment is of type 'application/x-zip', who are we to question that (in [add_attachment\(\)](#)). However, if a message is clearly 7bit ASCII with short lines and of type 'text' (text/plain or text/html or some other text-subtype), we do change the charset to US-ASCII and encoding to 7bit, to make the message just a little bit more readable for the receiver.

Finally, we use the mail() command to actually send the message. We add an additional parameter which should instruct sendmail to use the from-address also as the return path (if that is allowed and the webserver is a 'trusted user').

- **Uses** [Email::rfc5322_message_id\(\)](#)
- **Uses** [Email::rfc5322_address\(\)](#)

- Uses [Email::rfc2047_qstring\(\)](#)
- Uses [Email::is_7bit\(\)](#)

void function Email::set_header(\$name, [\$value = "]) [*line 259*]

Function Parameters:

- *string* **\$name** is the name of the header field
- *string* **\$value** is the contents of the header field

manually add a header to the mail message

This adds a header to the message headers. Possible candidates are

- 'Priority' with possible values (from RFC2156): "normal" | "non-urgent" | "urgent"
- 'Importance' with possible values (from RFC2156): "low" | "normal" | "high"

Note that the following headers may be overwritten in the course of constructing the message to send (see [send\(\)](#)):

- To: (depending on how mail() handles the first parameter internally)
- Subject: (depending on how mail() handles the second parameter internally)
- From:
- Reply-To:
- Cc:
- X-Mailer:
- Message-ID:
- MIME-Version:
- Content-Type:
- Content-Transfer-Encoding:

It is possible to use tricks such as using a different capitalisation to defeat this. (I said it was a simple class, didn't I?)

- **TODO** should we bring the Capi-Tali-Sation of \$name in line with the default capitalisation in the list above?

void function Email::set_mailfrom(\$addr, [\$name = "]) [*line 133*]

Function Parameters:

- *string* **\$addr** the address eg. 'webmaster@example.com'
- *string* **\$name** the name, eg. 'Exemplum Primary School'

record the address and the name for the From: header

Embedded CRs LFs "<" and ">" are removed from \$addr, and the result is stored, together with \$name.

void function Email::set_mailreplyto(\$addr, [\$name = "]) [line 149]

Function Parameters:

- *string* **\$addr** the address eg. 'info@example.com'
- *string* **\$name** the name, eg. 'Exemplum Primary School'

record the address and the name for the Reply-To: header

Embedded CRs LFs "<" and ">" are removed from \$addr, and the result is stored, together with \$name.

void function Email::set_mailto(\$addr, [\$name = "]) [line 165]

Function Parameters:

- *string* **\$addr** the address eg. 'hparkh@example.com'
- *string* **\$name** the name, eg. 'Helen Parkhurst'

record the address and the name for the To: header

Embedded CRs LFs "<" and ">" are removed from \$addr, and the result is stored, together with \$name.

void function Email::set_message(\$message, [\$mimetype = 'text/plain'], [\$charset = 'UTF-8'], [\$encoding = 'quoted-printable']) [line 203]

Function Parameters:

- *string* **\$message** content to send
- *string* **\$mimetype** type of the content, usually 'text/plain'

- *string* **\$charset** character set to use
- *string* **\$encoding** the desired encoding (defaults to quoted-printable because we expect text)

set the message

This simply stores the message body until it can be sent. Note that there can be just 1 message. It is assumed to be plain text. However, perhaps it could be a multipart/alternative (not tested).

void; function Email::set_subject(\$subject) [*line 119*]
Function Parameters:

- *string* **\$subject**

store the subject of the mail message

Embedded CRs and LFs in \$subject are removed and the result is stored until message send time.

Class FileManager

[*line 70*]

File Manager

This class implements the File Manager.

This class is also used to browse files and images from FCKEditor. Distinction is made via the \$job parameter in the constructor.

All the work is directed from the constructor, so it is enough to simply instantiate a new object and let the constructor do the work. The only thing needed is an output object (see [AdminOutput](#)).

- **Package** wascore

FileManager::\$areas

null|array = NULL [line 78]

- **Var** \$areas holds all area records (for future reference) or NULL if not yet set

FileManager::\$current_directory

string = [line 87]

- **Var** \$current_directory links to the session-variable that holds the current working directory

FileManager::\$ext_allow_browse

bool|array = FALSE [line 96]

- **Var** \$ext_allow_browse holds browsable filename extensions (lowercase), FALSE (none) or TRUE (all)

FileManager::\$ext_allow_upload

bool|array = FALSE [line 93]

- **Var** \$ext_allow_upload holds uploadable filename extensions (lowercase), FALSE (none) or TRUE (all)

FileManager::\$job

string = [line 75]

- **Var** \$job indicates how we are called (eg. as 'filemanager' or as 'filebrowser' or 'imagebrowser')

FileManager::\$output

object|null = NULL [line 72]

- **Var** \$output collects the html output

FileManager::\$show_thumbnails

bool = FALSE [line 99]

- **Var** \$show_thumbnails if TRUE we display files graphically (as a thumbnail), otherwise in table format

FileManager::\$sort

int = SORTBY_FILE_ASC [line 90]

- **Var** \$sort holds the current sort order in directory listings (default SORTBY_FILE_ASC)

FileManager::\$usergroups

null|array = NULL [line 81]

- **Var** \$usergroups holds all \$USER's group records (for future reference) or NULL if not yet set

FileManager::\$vpaths

array = array() [line 84]

- **Var** \$vpaths is a cache of virtual paths (see [vpath\(\)](#))

Constructor *void* function FileManager::FileManager(&\$output, [\$job = JOB_FILEMANAGER]) [*line 115*]

Function Parameters:

- *object* **&\$output** collects the html output
- *string* **\$job** indicates the mode: filemanager, filebrowser (FCKEditor) or imagebrowser (FCKEditor)

construct a FileManager object (called from /program/main_admin.php)

This initialises the FileManager, checks user permissions and finally dispatches the tasks. If the specified task is not recognised, the default task TASK_LIST_DIRECTORY is executed.

Note that many commands act on the directory contained in the SESSION-variable `current_directory`.

- **TODO** a nice filter for JOB_IMAGEBROWSER and also an alternative user interface for browsing/selecting images

bool/array function FileManager::allowed_extensions(\$allowed_extensions_list) [*line 2915*]

Function Parameters:

- *string* **\$allowed_extensions_list** comma-delimited string with allowable extensions (or empty string)

convert a comma-delimited list of allowable extensions to an array (or FALSE if none are allowed)

this converts the comma-delimited list of allowable filename extensions to an array with one element per allowable extension OR to a boolean with value FALSE if no allowable extensions are specified. The input is converted to lower case and also spaces and dots are removed (in anticipation of users entering the bare extension with the dot while we use the

bare extension here. Also, it appears very natural to specify a list including spaces (which we don't want), so there.

int function FileManager::cmp_entries_bydate_asc(\$a, \$b) [line 2234]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by mtime

Comparison between a file and a directory always shows directories first.

int function FileManager::cmp_entries_bydate_desc(\$a, \$b) [line 2260]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by date (descending)

Comparison between a file and a directory always shows directories first, nevermind that we are sorting in descending order.

int function FileManager::cmp_entries_byfile_asc(\$a, \$b) [line 2142]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by filename

Comparison between a file and a directory always shows directories first.

int function FileManager::cmp_entries_byfile_desc(\$a, \$b) [line 2163]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by filename (descending)

Comparison between a file and a directory always shows directories first, nevermind that we are sorting in descending order.

int function FileManager::cmp_entries_bysize_asc(\$a, \$b) [*line 2183*]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by size

Comparison between a file and a directory always shows directories first.

int function FileManager::cmp_entries_bysize_desc(\$a, \$b) [*line 2209*]

Function Parameters:

- *array \$a* first entry
- *array \$b* second entry

callback for comparing two directory entries by size (descending)

Comparison between a file and a directory always shows directories first, nevermind that we are sorting in descending order.

int function FileManager::cmp_groups(\$a, \$b) [*line 2099*]

Function Parameters:

- *array \$a* first array with groupdata (straight copy from database record)
- *array \$b* second array with groupdata (straight copy from database record)

callback for comparing two group records

This routine is used to order a list of groups by full_name, groupname.

bool function FileManager::delete_directory(\$path, \$directories, \$entries) [line 1569]

Function Parameters:

- *string* **\$path** the directory containing the directories to delete
- *array* **\$entries** list of subdirectories to delete, keyed by subdirectory name
- **\$directories**

workhorse function that actually removes directories

This routine first deletes the files "index.html" and "THUMBNAIL_PREFIX*" in the directory to remove and subsequently the directory itself. This is more or less the reverse of the mkdir function (see [task add_directory\(\)](#)) but with a twist: we consider any remaining thumbnails as trash and we will happily delete those without further ado.

If anything goes wrong, the routine returns FALSE and some details are written to the logfile. Note that if the directory somehow contains symlinks or devices or named pipes we bail out: we cannot handle those kinds of directory entries.

Note that the routine is able to delete an array of directories, even though it is currently called/used with only a single entry. We don't want to make it too easy to remove many directories at once (an attempt to protect the user against herself).

- **Uses \$CFG**

bool function FileManager::delete_files(\$path, \$entries) [line 1520]

Function Parameters:

- *string* **\$path** the directory containing the files to delete
- *array* **\$entries** list of files to delete

workhorse function that actually deletes files, and possibly the corresponding thumbnails

This routine deletes the files specified in the array \$entries from directory \$path. If a thumbnail-file exists (ie. a file with a similar name but with the THUMBNAIL_PREFIX prepended), it is deleted too.

- **Uses \$CFG**

array function FileManager::explode_path(\$path) [line 1711]

Function Parameters:

- *string* **\$path** the path to split

shorthand for splitting a path into an array with path components

This routine splits \$path into components. Path components are supposed to be delimited with a forward slash '/', but if somehow a backslash '\' is encountered, it is translated to a forward slash first. This means that it is impossible to have backslashes as part of a path name, even though the underlying filesystem would happily accept a component (filename or directoryname) with an embedded backslash.

string function FileManager::file_url(\$path) [line 2574]

Function Parameters:

- *string* **\$path** the name of the file including path

construct a url that links to a file via /file.php

This constructs a URL that links to a file, either

/file.php/path/to/file.txt

or

/file.php?file=/path/to/file.txt

depending on the global setting for proxy-friendly urls. Note that we try to make the link as short as possible, eg. by omitting the http:// part if possible (see \$CFG->www_short).

array function FileManager::get_dialogdef_add_files([\$num_files = 1]) [line 2288]

Function Parameters:

- ***int* \$num_files** the maximum number of file upload fields to add to the dialog (default 1)

construct a dialog definition for adding (uploading) files

this constructs an array which defines a file(s) upload dialog. Note that we make a subtle difference between a single-file upload and a multifile upload: I think it looks stupid to start numbering a list of files to upload when there is in fact only a list of exactly 1 file(s). The cost is minimal: two extra strings in the translation file.

array function FileManager::get_entries(\$path) [*line 1944*]

Function Parameters:

- ***string* \$path** the directory to list

generate a list of selected files and subdirectories in \$path

This creates an array containing a (filtered) listing of the directory \$path, keyed by filename. we items are suppressed:

- current directory '.'
- parent directory '..'
- index.html if it has size 0 (used to 'protect' directory against prying eyes)
- THUMBNAIL_PREFIX* the thumbnails of images
- symbolic links

- **Uses** \$USER;
- **Uses** \$CFG;

array function FileManager::get_entries_areas() [*line 1800*]

generate a list of (virtual) directories for areas the user can access

This generates a list of (virtual) area directories for which the user has access permissions. The list is ordered based on the sort order (in the areas table).

- **Uses** \$USER;
- **Uses** \$CFG;

array function FileManager::get_entries_groups() [line 1836]

generate a list of (virtual) directories for groups the user can access

This generates a list of (virtual) group directories for which the user has access permissions. The list is ordered by groupname.

- **Uses** \$USER;
- **Uses** \$CFG;

array function FileManager::get_entries_root() [line 1728]

generate a list of (virtual) directories at the root level

This generates a list of up to 4 'directories' which are equivalent to 'My Files', 'Areas', 'Groups' and 'Users'. Permissions and group memberships are taken into account, i.e. if a user has no group memberships (and is not an account manager), the 'Groups' directory is suppressed.

- **Uses** \$USER
- **Uses** \$CFG

array function FileManager::get_entries_users() [line 1884]

generate a list of (virtual) directories for users this user can access

This generates a list of (virtual) user directories for which this user has access permissions. The list is ordered by full name.

- **Uses** \$USER;
- **Uses** \$CFG;

bool function FileManager::has_allowed_extension(\$filename, &\$extensions) [*line 2886*]

Function Parameters:

- *string* **\$filename** the filename to examine
- *bool/array* **&\$extensions** array with allowable extensions or FALSE for none or TRUE for all

see if the filename extension is allowed

Note that an 'empty' extension could be acceptable.

string function FileManager::human_readable_size(\$size) [*line 2076*]

Function Parameters:

- *int* **\$size** value to convert

convert an integer filesize to a human readable form

This routine displays a file size in bytes, kilobytes, megabytes or gigabytes or bytes with space-delimited groups of 3 digits depending on the size. No decimals are used.

void function FileManager::make_thumbnail(\$directory, \$filename) [*line 2617*]

Function Parameters:

- *string* **\$directory** the working directory (relative to \$CFG->datadir)
- *string* **\$filename** the name of the image file (including extension if any)

try to create a thumbnail of the image in file \$filename (best effort)

this routine attempts to create a thumbnail of file \$filename. First we determine whether this is actually a graphics file and whether GD is available. Then we decide if scaling is needed at all. If the file is of one of the currently supported image formats, we load the file, resample it and write the resulting (smaller) image to a file which name is prepended with the thumbnail prefix.

Design considerations:

- we use a square box (of \$thumb_dimension x \$thumb_dimension) to fit the images
 - we do not create thumbnails for images smaller than that square box
 - the aspect ratio is preserved
 - we use default quality settings for write jpeg and png thumbnails
 - all errors are logged, successes are logged to LOG_DEBUG
 - this routine totally relies on GD
 - we try to extend our stay with set_time_limit() every time a thumbnail is created because image processing is quite time-consuming
- Note that this is a best effort: if something goes wrong, we do not try very hard to correct the error; creating a thumbnail is considered 'nice to have'. As a matter of fact we leave immediately on error (after cleaning up memory hogs, naturally).

Note that we only use GD to create thumbnails. I did consider Imagick, but eventually I decided against it because it appears that support for that extension requires PHP 5.1.3. Perhaps we can add support in a later version of the FileManager.

string function FileManager::sanitise_filetype(\$path, \$name, \$type) [line 2457]

Function Parameters:

- *string* **\$path** full path to the actual file (from \$_FILES[\$i]['tmp_name'])
- *string* **\$name** the requested name of the file to examine (from \$_FILES[\$i]['name'])
- *string* **\$type** the suggested filetype of the file (from \$_FILES[\$i]['type'])

try to make sure that the extension of file \$name makes sense or matches the actual filetype

this checks or changes the \$name of the file in line with the mimetype of the actual file (as established by get_mimetype()).

The reason to do this is to make it harder to 'smuggle in' files with deceptive filenames/extensions. Quite often the extension is used to determine the type of the file, even by browsers that should know better. By uploading a malicious .PDF using an innocuous extension like .TXT, a browser may be tricked into rendering that .PDF inline. By changing the extension from .TXT to .PDF we can mitigate that risk, at least a little bit. (People somehow trust an extension even though they should know better and file(1) says so...)

Strategy is as follows. If the mimetype based on the \$name matches the actual mimetype, we can simply allow the name provided.

If there is a difference, we try to find an extension that maps to the same mimetype as that of the actual file. IOW: we put more trust in the mimetype of the actual file than we do in the mimetype suggested by the extension.

void function FileManager::show_breadcrumbs(\$path) [*line 1347*]

Function Parameters:

- *string* **\$path** the directory path to show

display a clickable path to the directory \$path

dialog function FileManager::show_dialog_confirm_delete_directory(\$path, \$entries_to_delete) [*line 1480*]

Function Parameters:

- *string* **\$path** the working directory
- *array* **\$entries_to_delete** an array with directory entries identifying the files to delete keyed by name

show a dialog that ask the user to confirm the removal of a directory

Show the first directory from \$entries_to_delete and ask the user to confirm with [Delete] button or to cancel with [Cancel] button. The name of the directory to delete is part of the href rather than a POSTed field. We only allow a single directory to be removed at a time.

dialog function FileManager::show_dialog_confirm_delete_files(\$path, \$entries_to_delete) [*line 1426*]

Function Parameters:

- *string* **\$path** the working directory
- *array* **\$entries_to_delete** an array with directory entries identifying the files to delete keyed by name

show a dialog that ask the user to confirm a mass file delete

Show a list of files from \$entries_to_delete and ask the user to confirm with [Delete] button or to cancel with [Cancel] button. The names of the files to delete are communicated via hidden fields. Once the form is submitted the data is validated against the existing files in the directory \$path.

- Usedby [FileManager::task_remove_file\(\)](#)
- Usedby [FileManager::task_remove_multiple_files\(\)](#)

void function FileManager::show_directories(&\$entries, \$parent) [line 1299]

Function Parameters:

- *array* **&\$entries** ready to use data describing all subdirectories to show
- *bool/string* **\$parent** suppress link to parent if FALSE otherwise path of parent

output a simple list of directories (for navigation only)

This outputs a simple list of subdirectories based on information in the array \$entries. The subdirectories can not be deleted and no files or subdirectories can be added. (because this is either '/', 'Areas','Groups' or 'Users')

- Usedby [FileManager::show_list\(\)](#)
- Uses \$WAS_SCRIPT_NAME
- Uses \$USER
- Uses \$CFG

void function FileManager::show_directories_and_files(\$path, [\$show_thumbnails = TRUE]) [line 918]

Function Parameters:

- *string* **\$path** the directory to display
- *bool* **\$show_thumbnails** if TRUE files are displayed as thumbnails, table rows otherwise

display a list of subdirectories and files in directory \$path

This long routine displays the following items to the user

- (optional) navigation link to add (upload) a file
- (optional) navigation link to add (create) a directory
- a 4, 5 or 6 column table with
 - . navigation link to the parent directory

- . 0, 1 or more rows with delete and navigation links to subdirectories (if any)
- . 0, 1 or more rows with a checkbox and delete and preview links to files (if any)
- . (optional) a 'select all' checkbox
- (optional) Delete-button to mass-delete files

The table can be ordered in various ways: by name, by size and by date, either ascending or descending. Clicking the relevant column header yields another sort order. This toggles between ascending and descending. Default sort order is by name ascending.

The checkbox 'select all' works with Javascript in the most simple way: an ad-hoc script connected to the onclick attribute of the select all checkbox. However, the select all checkbox itself is rendered via Javascript. The effect is that this feature is only available if Javascript is enabled in the browser. If it isn't, no select all is visible so it can not distract the user. This is part of the attempt to make this CMS usable even without Javascript.

If the flag \$show_thumbnails is set we display file entries as thumbnails. This is done mostly to cater for the visual interactive selection of images from FCK Editor.

- **TODO** This routine is way too long, it should be split up into smaller subroutines
- **Usedby** [FileManager::show_list\(\)](#)
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

output function FileManager::show_file_as_thumbnail(\$directory, \$entry, \$delete_file, \$index, [\$m = ""]) [line 2770]
Function Parameters:

- *string* **\$directory** the current working directory (necessary to construct (full) paths)
- *array* **\$entry** information about the file to show, see [get_entries\(\)](#) for the format
- *bool* **\$delete_file** if TRUE, user is allowed to delete the file (used for generating delete icon)
- *int* **\$index** a counter used to generate a unique field name for the checkbox
- *string* **\$m** optional margin for better code readability

show a thumbnail of a single (image) file perhaps including clickable links for selection in FCK Editor

This constructs a single clickable image with either a selection of the file (for FCK Editor, in file/image browser mode) or a link to the file preview. If a file is not an image or otherwise no suitable thumbnail is found, a large question mark is displayed (unknown.gif). Otherwise the existing thumbnail is shown, maintaining the original aspect ratio. Either way the image is scaled to the currently specified thumbail dimension so the image fits the corresponding DIV-tag.

The strategy for finding a thumbnail is as follows: - is the file to show an image at all? If not, show unknown.gif - if the file zz_thumb_{filename.ext} exists, use that, otherwise - if not AND the original file is smaller than a thumbailm use the original file, otherwise - use 'unknown.gif' after all.

If the flag \$delete_file is set, we also generate a checkbox and a delete icon. This means that even in file/image browser mode files can be deleted by the user. In fact the file/image browser is basically the same old filemanager.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function FileManager::show_list(\$path) [*line 853*]

Function Parameters:

- *string* **\$path** the (virtual) path to the directory to list

display a list of directories and files in \$path

This yields a list of directories (for \$path is '/', '/areas', '/groups' or '/users') or a list of directories and files (\$path is anything else). In the latter case, if the current user has sufficient permissions, various additional links are added such as upload a file or create directory. The actual work is done in two separate workhorses.

- **Uses** [FileManager::show_directories_and_files\(\)](#)
- **Uses** [FileManager::show_directories\(\)](#)

void function FileManager::show_menu([\$current_path = "]) [line 1390]

Function Parameters:

- *string* **\$current_path** indicator for highlighting the current directory subtree

show a menu that is equivalent with the root directory

void function FileManager::sort_entries(&\$entries, \$sort, \$sortorder) [line 2119]

Function Parameters:

- *array* **&\$entries** array with directory entries
- *int* **\$sortorder**
- **\$sort**

sort directory entries

- **TODO** it is a pity I cannot reference `$this->sort` from within the 6 cmp-functions...

void function FileManager::task_add_directory() [line 623]

create a new subdirectory

This routine either shows a dialog where the user can specify the name of a new directory to add OR processes the dialog.

In case of directory name too short, already exists, etc. the user is returned to the dialog to try again. If all goes well the new directory is created and at the same time the empty file 'index.html' is created to "protect" the directory from prying eyes.

void function FileManager::task_add_file() [line 468]

add one or more new files to a directory

This routine either shows a dialog where the user can specify the names of one or more (maximum `$CFG->upload_max_files`) files OR processes the dialog.

Various checks are performed before the files are actually saved, e.g. checks for viruses (via ClamAV), resolve name clashes, allowed filetypes and extensions, etc. This is all done in a separate worker routine.

void function FileManager::task_change_directory() [line 239]

make another directory the current (working) directory and optionally change the sort order

This changes the current working directory to the user-supplied path (after thorough validation, naturally). The new current directory is stored in `$this->current_directory` and via that reference in the `$_SESSION` array, for future reference. If a valid directory was specified, we also take a look at the optional sort order parameter and set the sort order of the directory listing accordingly.

After (perhaps) changing the current directory, and perhaps changing the sort order, the contents of that directory is displayed via [task_list_directory\(\)](#).

void function FileManager::task_list_directory() [line 214]

show a directory listing of the current working directory and links to add/delete files/directories etc.

This is the main routine to show a list of subdirectories and files in the current working directory (`$_SESSION['current_directory']`).

void function FileManager::task_preview_file() [line 258]

preview a file via file.php

After validation of the specified path, the user is redirected to [file.php](#) in order to show the selected file.

void function FileManager::task_remove_directory() [line 394]

show a confirmation dialog for removing a single directory OR actually removes a directory

This shows a confirmation dialog for removing of a single directory OR actually removes a directory.

void function FileManager::task_remove_file() [line 352]

show a confirmation dialog for deleting a single file

This shows a confirmation dialog for deletion of a single file. We reuse the code for deletion of multiple files, see [task_remove_multiple_files\(\)](#).

- **Uses** [FileManager::show_dialog_confirm_delete_files\(\)](#)

void function FileManager::task_remove_multiple_files() [*line 279*]

show confirmation dialog for multiple file delete OR perform actual file delete

this routine either shows a list of files to be deleted, asking the user for confirmation or actually deletes the specified files if the user did confirm the delete. We bail out if the user pressed the cancel button in the confirmation dialog. The real work is done in workhorse routines in order to combine the single-file-delete and the batch-delete into a single confirmation routine. For actual deletion, however, we always return here and not in the single file delete (see `{\$link task_remove_file\(\)}`).

- **Uses** [FileManager::show_dialog_confirm_delete_files\(\)](#)

string function FileManager::unique_filename(\$directory, \$name) [*line 2530*]

Function Parameters:

- *string* **\$directory** the target directory for the file upload
- *string* **\$name** the (sanitised) name of the file

construct a unique filename taking existing files into account

this constructs a filename that is unique within the target directory. If a file of the name \$name already exists, a new name is constructed from the basename of \$name, an integer sequence number and the extension of \$name.

There is no guarantee that this name will stay unique between the moment we test for it and the moment the file is actually moved into place (a classical race condition). However, the chance that this will be happening is small enough I guess.

- **TODO** Should we take care of the race condition in this routine? Should we already create an empty file or is that clutter?

- **Uses \$CFG**

string|bool function FileManager::valid_path(\$path) [*line 758*]

Function Parameters:

- *string* **\$path** the path (file or directory) to check, relative to \$CFG->datadir

access control and validation for selected directory or file

This routine checks to see if the current user has access to the specified file or directory. If not, FALSE is returned, otherwise the valid path is returned, with a starting slash. If \$path doesn't start with a slash a slash is assumed nevertheless. The path is relative to \$CFG->datadir.

It is not allowed to reference parent directories in the path. This prevents tricks like '../../etc/passwd' (leaking the system passwd file) or '/users/foo/../bar' (access to bar's userdata with permissions for just foo's files). Furthermore, symbolic links are NOT acceptable as part of the path, i.e. symlinks like '/users/foo/etc -> /etc' or '/users/foo/passwd -> /etc/passwd' are considered invalid.

Required permissions for access are:

- areas: \$USER must have the admin_pagemanager permissions for that area.
- groups: \$USER must be a member of that group OR \$USER must have access to the Account Manager.
- users: \$USER must be that user OR \$USER must have access to the Account Manager.

- **TODO** the check on '../' is inconclusive if the \$path is encoded in UTF-8: the overlong sequence 2F C0 AE 2E 2F eventually yields 2F 2E 2E 2F or '../'. Reference: RFC3629 section 10.
- **Uses \$USER**
- **Uses \$CFG**

int function FileManager::virusscan(\$path, [\$name = ""]) [*line 2351*]

Function Parameters:

- *string* **\$path** the path of the file to scan
- *string* **\$name** the name of the file as provided by the uploader (from \$_FILES)

scan a file for viruses

this scans \$path for viruses, returns 0 if file considered clean, 1 for infected file, or 2 if something else went wrong.

If the flag \$CFG->clamscan_mandatory is set, we consider the file infected if we are not able to run the virus scanner (better safe than sorry). However, if no virusscanner is configured at all (\$CFG->clamscan_path is empty), we indicate a 'clean' file even though we did not scan it. Rationale: it doesn't make sense to make scanning mandatory and at the same time NOT configuring a scanner at all.

If scanning succeeds and a virus is found we send an alert to the website owner address (or the reply-to-address) immediately. Furthermore everything is logged.

- **TODO** This routine is quite *nix-centric. I'm not sure how this would work other server platforms. Should we do something about that?
- **TODO** maybe use MIME for sending alert if not 7bit message?
- **Uses** \$USER
- **Uses** \$CFG

string function FileManager::vname(\$path) [line 1640]

Function Parameters:

- *string* **\$path** the path to examine

construct the (possibly translated) name of the last directory in the path

This examines \$path and returns a string with the last directory component. There are a few special cases:

- the empty string indicates the root directory
 - /users/<userpath> maps to a (translated) string t('filemanager_personal')
 - /areas maps to a (translated) string t('filemanager_areas')
 - /groups maps to a (translated) string t('filemanager_groups')
 - /users maps to a (translated) string t('filemanager_users')
- All other variations yield the last component in the list of components.

string function FileManager::vpath(\$path) [*line 2009*]

Function Parameters:

- *string* **\$path** the path to translate

translate a path to the corresponding virtual path

This translates a path like '/users/webmaster/foo' into 'My Files/foo' and '' into 'All Files', etc. The result of the translation is cached in \$this->vpaths, for future reference.

Class GroupManager

[*line 37*]

Group management

- **Package** wascore
- **TODO** Perhaps this class should be merged with the UserManager class because there is a lot of overlap. Mmmmm.... maybe in a future refactoring operation.

GroupManager::\$group_capacity_records

array|null = NULL [*line 42*]

- **Var** caches the list of group-capacity-combinations

GroupManager::\$output

object|null = NULL [*line 39*]

- **Var** collects the html output

GroupManager::\$show_parent_menu

bool = FALSE [line 45]

- **Var** if TRUE the calling routing is allowed to use the menu area (e.g. show account mgr menu)

Constructor *void* function GroupManager::GroupManager(&\$output) *[line 53]*

Function Parameters:

- *object* **&\$output** collects the html output

construct a GroupManager object

This initialises the GroupManager and also dispatches the task to do.

bool function GroupManager::acl_delete(\$acl) *[line 1679]*

Function Parameters:

- *int/array* **\$acl** the key(s) to the ACL(s) to delete

remove all records relating to 1 or more acl_id's from various acl-tables

this bluntly removes all records from the various acls* tables for the specified acl_id's. Whenever there's an error deleting records, the routine bails out immediately and returns FALSE. If all goes well, TRUE is returned. Any errors are logged, success is logged to DEBUG-log.

- **TODO** should this routine be moved to an acl-object? Hmmmm....

void function GroupManager::add_group_capacity(\$group_id, \$capacity_code, \$sort_order) *[line 1711]*

Function Parameters:

- **\$group_id**
- **\$capacity_code**
- **\$sort_order**

array|bool function GroupManager::areas_expand_collapse(\$areas_open, \$area_id) [line 1824]

Function Parameters:

- *array|bool* **\$areas_open** current state of indicator(s) for 'open' and 'closed' areas
- *int|null* **\$area_id** the area to expand/collapse or NULL if nothing needs to be done

manipulate the current state if indicator(s) for 'open' and 'closed' areas

this manipulates the current state of 'open' and 'closed' areas in \$areas_open. If \$area_id is NULL, we don't have to do anything but simply return the current state. If \$area_id is 0 (zero), we need to toggle all areas at once (area_id = 0 implies the site level toggle) If \$area_id is an integer, it is assumed to be a valid area_id and that area should be toggled.

array function GroupManager::a_params([\$task = NULL], [\$group_id = NULL], [\$capacity_code = NULL], [\$module_id = NULL]) [line 1559]

Function Parameters:

- *string|null* **\$task** the next task to do or NULL if none
- *int|null* **\$group_id** the group of interest or NULL if none
- *int|null* **\$capacity_code** the capacity of interest or NULL if none
- *int|null* **\$module_id** the module of interest or NULL if none

shorthand for the anchor parameters that lead to the group manager

void function GroupManager::calc_acl_id(\$group_id, \$capacity_code) [line 1637]

Function Parameters:

- **\$group_id**

- **\$capacity_code**

void function GroupManager::capacity_admin() [line 1024]

show a dialog for modifying admin permissions for a group/capacity

- **Uses \$WAS_SCRIPT_NAME**
- **Uses \$CFG**

void function GroupManager::capacity_intranet() [line 987]

show a dialog for modifying intranet permissions for a group/capacity

- **Uses \$WAS_SCRIPT_NAME**
- **Uses \$CFG**

void function GroupManager::capacity_overview() [line 821]

display an overview of all members of a group with a particular capacity

this constructs a clickable list of users that are associated with a particular combination of group_id and capacity_code. The name of the user is a link to the usermanager for that user. Users are sorted by name.

- **Uses \$WAS_SCRIPT_NAME**
- **Uses \$DB**

void function GroupManager::capacity_pagemanager() [line 1061]

show a dialog for modifying page manager permissions for a group/capacity

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG

void function GroupManager::capacity_save() [line 877]
save data from a dialog for a group/capacity

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG

bool function GroupManager::delete_group_capacities_acls(\$group_id) [line 1767]
Function Parameters:

- *int* **\$group_id** the group to delete

actually remove a group and all associated data

this actually deletes the group \$group_id and associated data, in the following order: First all acls associated with the group-capacities are deleted. If that worked, we delete the group-capacity records. If that worked, we delete the group record itself.

- **TODO** should we also require the user to delete any files associated with the area before we even consider deleting it? Or is it OK to leave the files and still delete the area. We do require that nodes are removed from the area, but that is mainly because of maintaining referential integrity. Mmmmm... Maybe that applies to the files as well, especially in a private area. Food for thought.
- **TODO** since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

array function GroupManager::get_dialogdef_add_group() [line 1359]
construct the add group dialog

array/bool function GroupManager::get_dialogdef_edit_group(\$group_id) [*line 1418*]

Function Parameters:

- *int* **\$group_id** the group that will be edited

construct the edit group dialog

void function GroupManager::get_groupname(\$group_id) [*line 1631*]

Function Parameters:

- **\$group_id**

void function GroupManager::get_group_capacity_names(\$group_id, [\$capacity_code = 0]) [*line 1661*]

Function Parameters:

- **\$group_id**
- **\$capacity_code**

array/bool function GroupManager::get_group_capacity_records([\$force = FALSE]) [*line 1531*]

Function Parameters:

- **\$force**

return an array of group-capacity records (possibly buffered)

- **Uses \$DB**

string function GroupManager::get_icon_delete(\$group_id) [*line 1584*]

Function Parameters:

- *int* **\$group_id** the group to delete

construct a clickable icon to delete this group

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

string function GroupManager::get_icon_edit(\$group_id) [*line 1611*]

Function Parameters:

- *int* **\$group_id** the group to edit

construct a clickable icon to edit the properties of this group

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function GroupManager::get_options_capacities() [*line 1513*]

void function GroupManager::groups_overview() [*line 163*]

display list of existing groups and an option to add a group

this constructs the heart of the group manager: a link to add a group followed by a list of links for all existing groups and additional links per capacity per group.

This list of groups is ordered as follows. All active groups come first, the inactive groups follow. The sort order is based on the (short) name of the group.

Example:

Add a group

[D] [E] faculty (Member, Principal)

[D] [E] grade12 (Pupil, Teacher)

...

[D] [E] zebra (Member, Project lead)

[D] [E] aardvark (inactive)

[D] [E] grade45 (inactive)

Note that both the links '[E]' and 'faculty' lead to edit of group properties. The links 'Member' and 'Principal' lead to the group-capacity overview screen. The link '[D]' leads to a group delete confirmation screen.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function GroupManager::group_add() [line 282]

present 'add group' dialog where the user can enter minimal properties for a new group

this displays a dialog where the user can enter the minimal necessary properties of a new group. These properties are:

- name (e.g. 'grade7')
- full name (e.g. 'Pupils of grade 7')
- the active flag
- the allowable capacities for this group (e.g. 'Pupil' and 'Teacher')

Other properties (if any) will be set to default values and can be edited later on by editing the group.

The new group is saved via performing the task TASK_GROUP_SAVE_NEW

- **Uses** \$WAS_SCRIPT_NAME

void function GroupManager::group_delete() [line 712]

delete a group after confirmation

this either presents a confirmation dialog to the user OR deletes a group with associated capacities and acls.

Note that this routine could have been split into two routines, with the first one displaying the confirmation dialog and the second one 'saving the changes'. However, I think it is counter-

intuitive to perform a deletion of data under the name of 'saving'. So, I decided to use the same routine for both displaying the dialog and acting on the dialog.

- **TODO** should we also require the user to delete any files associated with the group before we even consider deleting it? Or is it OK to leave the files and still delete the group. Food for thought.
- **TODO** since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

void function GroupManager::group_edit() [line 448]

show a dialog with the basic properties of a group

- **Uses** \$WAS_SCRIPT_NAME

void function GroupManager::group_save() [line 479]

save an edited group to the database, including adding/modifying/deleting group/capacity-records

Note: no error checking when inserting new capacity because we more or less know that that capacity does not exist already or it would have been in the array already. (But what if there are more than GROUPMANAGER_MAX_CAPACITIES in the database? Mmmm....

- **Uses** \$WAS_SCRIPT_NAME;

data function GroupManager::group_savenew() [line 316]

save a new group to the database

this saves a new group to the database. This quite a complex task because of the number of tables involved.

First we have the table 'groups' which stores the basic group information. Then there is the table 'groups_capacities'. For every combination of group and capacity requested by the user

a record must be added to this table. Then there is also a separate acl for every group_capacity, so there.

The strategy should be something like this. new_group_id = insert_new_group_into_groups()
for all GROUPMANAGER_MAX_CAPACITIES do if capacity != CAPACITY_NONE
&& capacity_not_added_yet() prepare_new_acl_record(); new_acl_id =
insert_new_acl_in_acls(); prepare_new_groups_capacities_record();
insert_new_group_capacity_in_table()

- **TODO** maybe we should find a more elegant way to check a field for uniqueness
- **TODO** should we delete the datadirectory if something goes wrong?

bool function GroupManager::has_job_permission(\$group_id, \$capacity_code, \$job) [*line 1242*]

Function Parameters:

- *int* **\$group_id** group to check
- *int* **\$capacity_code** capacity of this group to check
- *int* **\$job** job a bitmask indicating a particular job

determine whether a group/capacity has permissions for a particular job

this determines whether this group/capacity has permissions to access the specified job, e.g. do they have access to the page manager. If so, we can display the menu option, otherwise we can suppress it and keep the menu clean(er).

void function GroupManager::show_breadcrumbs_addgroup() [*line 1329*]

display breadcrumb trail that leads to the add new group dialog

- **Uses** \$WAS_SCRIPT_NAME;

void function GroupManager::show_breadcrumbs_group() [*line 1257*]

display breadcrumb trail that leads to groups overview screen

- **Uses** \$WAS_SCRIPT_NAME;

void function GroupManager::show_breadcrumbs_groupcapacity(\$group_id, \$capacity_code) [line 1137]

Function Parameters:

- **\$group_id**
- **\$capacity_code**

display breadcrumb trail that leads to group capacity overview screen

- **Uses** \$WAS_SCRIPT_NAME;

void function GroupManager::show_menu_group(\$group_id, [\$current_task = NULL], [\$current_capacity_code = NULL]) [line 1282]

Function Parameters:

- **\$group_id**
- **\$current_task**
- **\$current_capacity_code**

show a menu for a group including links to the group's capacity overview screens

- **Uses** \$WAS_SCRIPT_NAME;

void function GroupManager::show_menu_groupcapacity(\$group_id, \$capacity_code, [\$current_task = NULL],

[\$current_module_id = NULL]) *[line 1167]*

Function Parameters:

- **\$group_id**
- **\$capacity_code**
- **\$current_task**
- **\$current_module_id**

void function GroupManager::show_parent_menu() *[line 122]*

void function GroupManager::valid_group_capacity(\$group_id, \$capacity_code) *[line 1650]*

Function Parameters:

- **\$group_id**
- **\$capacity_code**

Class Language

[line 33]

Translations of messages in different languages

- **Package** wascore

Language::\$default_domain

string = *[line 41]*

- **Var** \$default_domain the text domain to use if none is specified

Language::\$languages

array = array() [line 38]

- **Var** \$languages a cached list of all language records

Language::\$phrases

array = array() [line 35]

- **Var** \$phrases a cache of translated phrases

Constructor *void* function Language::Language([\$default_domain = "]) [line 51]

Function Parameters:

- *string* **\$default_domain** used when no domain is specified when requesting a translation

constructor

Set up the instance of this class. If no default domain is specified, 'was' is used. We always read the current list of all languages into core, for future reference.

array function Language::get_active_language_names() [line 109]

return an array with active languages and language names

this returns an array with language_key => language_name pairs, one entry per active language, ordered by language name. This array can be used in language picklists or to translate a language key to readable form. Note that we use the name of a language expressed in the language itself.

string function Language::get_current_language() [line 144]

determine the default language to use for translation of phrases

This routine determines which language to use for prompts and messages if not specified explicitly in calls to \$this->get_phrase(). There are various ways in which a language can be determined. Here's the list, in order of significance:

- \$_GET['language']

- `$_SESSION['language_key']`
- `$USER->language_key`
- `$CFG->language_key`
- constant value 'en' (the native language)

Note that all languages are validated against the list of valid and active languages as collected in `$this->languages`. If a language is NOT valid, the next test is tried. If all else fails we return 'en' for English, which is the native language and which should always be valid.

- **Uses** `$USER`;
- **Uses** `$CFG`

array function `Language::get_filenames_to_try($full_domain, $location_hint, $language_key)` [line 453]

Function Parameters:

- *string* **`$full_domain`** indicates the text domain including optional module/theme/addon prefix
- *string* **`$location_hint`** hints at a location of language file(s)
- *string* **`$language_key`** target language

calculate an ordered list of filenames to try for translation of phrases

WAS uses a separate language file for every text domain; basically the name of the text domain is the name of the file without the .php-extension. However, in order to prevent name clashes, modules and themes and addons have their own prefix: 'm_' for modules and 't_' for themes and 'a_' for addons.

The language translations for the installer are based on more or less the same trick: the prefix 'i_' identifies files in the directory `/program/install/languages`.

This trick with prefixing leads to the following search orders for generic phrases and module-, theme- and addon-specific phrases.

Example	1:	phrases	with	<code>\$domain='login':</code>	<code>{ \$CFG-</code>
		<code>>progdire}/languages/{ \$language_key}/login.php</code>			<code>{ \$CFG-</code>
		<code>>datadire}/languages/{ \$language_key}/login.php</code>			

Example	2:	phrases	with	<code>\$domain='m_guestbook':</code>	<code>{ \$CFG-</code>
		<code>>progdire}/modules/guestbook/languages/{ \$language_key}/guestbook.php</code>			<code>{ \$CFG-</code>
		<code>>progdire}/languages/{ \$language_key}/m_guestbook.php</code>			<code>{ \$CFG-</code>
		<code>>datadire}/languages/{ \$language_key}/m_guestbook.php</code>			

Example 3: phrases with `$domain='login'` and a hint in `$location_hint:`
`{ $location_hint } { $language_key } /login.php` `{ $CFG-`
`>datadir}/languages/{ $language_key } /login.php`

Example 4: phrases with `$domain='m_guestbook'` and a hint in `$location_hint:`
`{ $location_hint } { $language_key } /guestbook.php`
`{ $location_hint } { $language_key } /m_guestbook.php` `{ $CFG-`
`>datadir}/languages/{ $language_key } /m_guestbook.php`

Example 5: phrases with `$domain='i_demodata':` `{ $CFG-`
`>progdire}/install/languages/{ $language_key } /demodata.php` `{ $CFG-`
`>datadir}/languages/{ $language_key } /i_demodata.php`

- **Uses** `$CFG`

array function `Language::get_languages_to_try($language_key)` [line 395]
Function Parameters:

- *string* **`$language_key`** language of which to find all parents

calculate a list of possible languages and parent-languages to try for translations

This constructs an array with all ancestors (=parent languages) of `$language_key` and English if that language was not yet added.

string function `Language::get_phrase($phrase_key, [$full_domain = "], [$replace = "], [$location_hint = "], [$language_key = "])` [line 317]
Function Parameters:

- *string* **`$phrase_key`** indicates the phrase that needs to be translated
- *string* **`$full_domain`** (optional) indicates the text domain (perhaps with a prefix)
- *array* **`$replace`** (optional) an assoc array with key-value-pairs to insert into the translation
- *string* **`$location_hint`** (optional) hints at a directory location of language files
- *string* **`$language_key`** (optional) target language

translation of phrases via a phrase key

This routine looks up the text associated with the phrase key. If no domain is specified, the domain 'was' is tried. If no valid language is specified, the current language is used. If a location hint is specified, we trust the caller knows best where to look and we try locating a translations file in that directory location first.

Note that phrases in a particular language which are found later in the search overwrite the phrases found earlier. These additional phrases (dubbed 'dialect' or 'userdefined translations') can be used to overwrite or correct existing standard ('official') translations. These dialect phrases can be stored in a file in the languages subdirectory of the data directory (writable for the web server but hopefully outside the document root) and/or in the table 'phrases' in the database.

Whether these dialect phrases are actually fetched from disk or database depends on the configuration of the language, via the boolean fields 'dialect_in_database' and 'dialect_in_file'; if the corresponding switch is not TRUE, we don't even bother to go and look, which saves time.

Finally, if a particular phrase is not found in the requested language, we recursively try the parent language(s) of the requested language until there are no more parents. After that, we go for the 'en' translation. If that fails too, we return the phrase_key itself, sandwiched between the strings '(lang) ' and ' (/lang)', where 'lang' is the requested language code. Of course this should not happen if all translations are correct. (Famous last words...)

If a translation is found, we replace all occurrences of the keys in the array 'replace' in the translation with the corresponding values. This is done via a simple search and replace rather than a printf()-like way or (shudder) with complicated regex'es.

Note that we store search results in the array \$this->phrases so we can re-use those phrases in a next call. We cache the results on a per-domain basis, based on the assumption that after the first phrase in a particular domain is requested, it is likely that more phrases in the same domain will be requested.

Note that the resulting phrases are cached using the original language as the key (in \$this->phrases). This means that if a phrase in say 'de' or 'fr' was not found and 'en' was used instead, the English phrases are cached in the 'de' or 'fr' branch of the static array. This saves us time on the next call because we then use the phrases in the substitute language right away instead of going to look everywhere everytime.

Translations are fetched in such a way that the user-defined translations ('dialect') prevail over the system-defined ('official') translations. However, attempts to look for a phrase in a parent language (or 'en') only add the missing translations, preserving the translations in this full_domain that were already found. Quick illustration with Dutch (nl) and English (en): search order is: \$nl_database, \$nl_userfile, \$nl_system, \$en_database, \$en_userfile, \$en_system. The 'en' translations are only used if no corresponding Dutch translation is found. However, the English 'dialect' prevails over the English 'system' translation.

Examples of typical use of this routine:

```
echo $LANGUAGE->get_phrase('username');  
display the phrase from 'was.php' in the current language
```

```
echo $LANGUAGE->get_phrase('username','login');  
display the phrase from 'login.php' in the current language
```

```
echo $LANGUAGE->get_phrase('welcome','',array('{USERNAME}' => $USER->username));  
display the phrase from was.php in the default language, substituting the variable '{USERNAME}'.
```

- **TODO** should we return an error for an invalid specific language?
- **Uses** \$LANGUAGES

array function Language::get_phrases_from_database(\$full_domain, \$language_key) [*line 216*]

Function Parameters:

- *string* **\$full_domain** text domain to look for
- *string* **\$language_key** the language to look for

retrieve phrases from the database for specified domain and language

array function Language::get_phrases_from_file(\$filename) [*line 201*]

Function Parameters:

- *string* **\$filename** which language file to include

return the \$string array after including a file

This includes the specified language file \$filename (if it exists) and returns the array \$string. This assumes that filename actually consists of lines like this:

```
...  
$string['key_of_a_phrase'] = 'content of this phrase';  
$string['key_with_variable'] = 'Hello, {USERNAME}.';  
...
```

Because the file is included within the context of this function, the contents are added to the local array \$string rather than some global array. This is a feature.

Note that the included file MUST name the array '\$string' because otherwise this function will return an empty array. This means that any 'old' Site@School language files must be manipulated before they can be re-used. I'd consider this a feature too.

void function Language::reset_cache([\$language_key = "], [\$full_domain = "]) [line 500]

Function Parameters:

- *string \$language_key* the language
- *string \$full_domain* the language domain

remove selected entries (per language+domain, per language, or all) from cache

array function Language::retrieve_languages([\$force_reread = FALSE]) [line 69]

Function Parameters:

- *bool \$force_reread* if TRUE we always go to the database, else we try the cached version first

retrieve an array with all active languages from the database

This reads all languages from the database. If there's nothing there, we still return an array with a single element for the English language 'en', because 'en' is the native language of this program. If the language 'en' was not found, we still add it to the array. The resulting array is usually sorted by language name.

Class Module

[line 33]

Methods to access properties of a module

- **Package** wascore

Module::\$module_record

mixed = NULL [line 35]

Module::\$node_id

mixed = NULL [line 38]

Constructor *void* function Module::Module([\$module_record = NULL], [\$node_id = NULL]) [line 45]

Function Parameters:

- *array|null* **\$module_record** cached from modules table in database
- *int|null* **\$node_id** #node_id the node linked to this module

array function Module::get_breadcrumb_anchors(\$node_id) [line 64]

Function Parameters:

- *int* **\$node_id** identifies the node

get additional breadcrumb trail

string function Module::get_content(\$node_id) [line 55]

Function Parameters:

- *int* **\$node_id** identifies the node

get the actual content for node \$node_id

Class Node

[line 30]

/program/lib/node.class.php - taking care of nodes

This file defines a class for dealing with nodes.

- **Package** wascore
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: node.class.php,v 1.1.1.1 2011-02-01 13:00:19 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** we probably need to get rid of this file because it is not used (2010-12-07/PF)
- **License** [GNU AGPLv3+Additional Terms](#)

Node::\$node_exists

mixed = FALSE [*line 32*]

Node::\$node_path

mixed = array() [*line 34*]

Constructor *void* function Node::Node([\$requested_node = NULL], [\$requested_area = NULL]) [*line 36*]

Function Parameters:

- **\$requested_node**
- **\$requested_area**

bool/array function Node::calculate_node(\$node, \$area, &\$node_path) [*line 111*]

Function Parameters:

- *integer|null* **\$node** the requested node number or null if none specified
- *integer|null* **\$area** the requested area number or null if none specified
- *array* **&\$node_path** this referenced variable receives the (valid) path to the node to show

determine which node should be displayed based on user's request

In total there are 4 cases for node n and area a:

1. n defined, a defined
return n if it is visible and it matches with a and a is visible, otherwise FALSE

2. n defined, a undefined return n if n is visible and the corresponding a is visible, otherwise FALSE

3. n undefined, a defined return the default n in the specified a if a is active or FALSE if not found

4. n undefined, a undefined return the default n in the default a or FALSE if no active default area exists

Cases 1 and 2 might lead to a node that is not available (ie expired or under embargo). In order to not 'give away' information about the possible existence of the requested node, we simply return FALSE, indicating that the node was not found.

There is a potential problem with a request for a node that is of type 'section' (i.e. not a page that eventually yields actual content). The problem is that in that case we should look for a valid page-type node in that section-type node. The question is whether we should look for a default page in a (default) subsection or not (nested default). Currently the strategy is to look for the first suitable page if available, and otherwise descend into the first suitable subsection and try again repeatedly until we find a page or we reach the end of the tree.

Note: As a side effect of this function, the path from the area root to the final page is calculated. This navigation path is returned in the variable reference `$node_path`.

Calculation of this navigation path is necessary in order to determine the visibility of a node; if a section-type node higher in the tree is not available, all underlying nodes should also not be available. IOW: not only do we have to check the actual node, we need to check all parent nodes too.

The array returned in `$node_path` is keyed with the `node_id`, i.e. the array element `$node_path[$node_id]` yields the database record for node `$node_id`.

Note that we check the node embargo date and the expiry date in two separate expressions rather than using the 'BETWEEN low AND high' because you never can be sure if it is inclusive low/high or not and the database might assume the low is always smaller than high.

- **TODO** refactor into two different functions: one for specified node, other for unspecified node

void function Node::exists() [line 46]

void function Node::get_area_id() [line 54]

void function Node::get_node_path() [line 50]

Class PageManager

[line 79]

Page Manager

This class implements the Page Manager.

All the work is directed from the constructor, so it is enough to simply instantiate a new object and let the constructor do the work. The only thing needed is an output object (see [AdminOutput](#)).

- **Package** wascore

PageManager::\$areas

null|array = NULL [line 84]

- **Var** \$areas holds all area records (for future reference) or NULL if not yet set

PageManager::\$area_id

null|int = NULL [line 90]

- **Var** \$area_id indicates which tree is stored in \$this-tree, or NULL if none yet

PageManager::\$output

object|null = NULL [line 81]

- **Var** \$output collects the html output

PageManager::\$tree

null|array = NULL [line 87]

- **Var \$tree** holds the complete tree for area `$this->area_id` or NULL if not yet set

Constructor *void* function PageManager::PageManager(&\$output) *[line 111]*

Function Parameters:

- *object* **&\$output** collects the html output

construct a PageManager object (called from /program/main_admin.php)

This initialises the PageManager, checks user permissions and finally dispatches the tasks. If the specified task is not recognised, the default task TASK_TREEVIEW is executed.

Note that almost all commands act on the area contained in the SESSION-variable `current_area_id`. Also, we almost always need the tree of nodes in that area, so we read it once, `_before_` dispatching the task at hand. This means that the current tree in the current area is ALWAYS available. This means that none of the other routines should have to worry about which area or reading the tree; this information is already available in `$this->area_id` and `$this->tree`.

void function PageManager::build_cached_tree(\$area_id, [\$force = FALSE]) *[line 4004]*

Function Parameters:

- *int* **\$area_id** indicates which area to buffer if not already buffered
- *bool* **\$force** re-read of the tree for area `$area_id`

construct \$this->tree for future reference

this constructs the tree of the area `$area_id` so all other routines can simply use that tree instead of passing it around again and again via function arguments.

int function PageManager::calculate_new_sort_order(&\$tree, \$area_id, \$parent_id, [\$at_end = FALSE]) *[line 3821]*

Function Parameters:

- *array* **&\$tree** reference to the tree in area \$area_id
- *int* **\$area_id** the area to look at
- *int* **\$parent_id** the section where we need to make room (where a node is added/inserted)
- *bool* **\$at_end** if TRUE a new node is added at the end, otherwise it is inserted at the beginning

calculate a new sort order and at the same time make room for a node

this is used to calculate a new sort order number for a node that will be added to section \$parent_id in area \$area_id. Note that this could be another area than the current working area. The reference to the tree is necessary; we can't simply use \$this->tree and \$this->area_id.

Depending on the flag \$at_end the node is added at the end of the section or at the beginning. In the latter case, the new sort order number is always 10 and all the existing nodes are renumbered in such a way that the second node in the section (originally the first one) gets sort order 20. By not using consecutive numbers it is possible to 'insert' nodes without touching anything. This is not used but it does no harm to have a sort order in steps of 10 instead of 1. (I think the database doesn't care much when executing/interpreting the ORDER BY clause).

Note that this routine not only calculates a sort order but it also manipulates the database and moves other nodes in the section around in order to make room.

Note that the feature \$at_end is currently not used at all.

- **Uses \$DB**

int function PageManager::calculate_updated_sort_order(\$node_id, \$after_id) [*line 3909*]

Function Parameters:

- *int* **\$after_id** the node AFTER which \$node_id should be sorted (0 means: first in the section)
- **\$node_id**

calculate an updated sort order and also make space in the section for moving the node around

this calculates a new sort order for node node_id; the effect should be that node_id will

sort AFTER node after_id. If after_id is 0 then node_id should become the first in the section.

Note that this routine not only calculates a sort order but it also manipulates the database and moves other nodes in the section around in order to make room.

There are several different cases possible: a. \$after_id == 0 b. sort_order(\$after_id) < sort_order(\$node_id) c. sort_order(\$node_id) < sort_order(\$after_id) d. \$after_id is the last node in this section e. \$node_id == \$after_id

Case e. should not happen but if it did it would yield a no-op. Case d is very similar to case c, so much even that both cases can be combined to just one.

Strategy for case a. \$old_sort_order = sort_order(\$node_id); \$new_sort_order = sort_order(first_child(parent_section(\$node_id))) \$delta = \$old_sort_order - sort_order(prev(\$node_id)) SET \$sort_order += \$delta WHERE \$new_sort_order <= sort_order(node) <= \$old_sort_order

In other words: node_id gets the sort_order value from the first node in the section, all nodes from the first upto position where node_id was originally move 'up' in such a way that the last in that range will end up with the sort order that node_id had originally.

Strategy for case b. \$old_sort_order = sort_order(\$node_id) \$new_sort_order = sort_order(next(\$after_id)) \$delta = \$old_sort_order - sort_order(prev(\$node_id)) SET \$sort_order += \$delta WHERE \$new_sort_order <= sort_order(node) <= \$old_sort_order

Note that a and b are also quite similar.

Strategy for case c. (and d.) \$old_sort_order = sort_order(\$node_id) \$new_sort_order = sort_order(\$after_id) \$delta = \$old_sort_order - sort_order(next(\$node_id)) (note that this is a negative value) SET \$sort_order += \$delta WHERE \$old_sort_order <= sort_order(node) <= \$new_sort_order

By mass-updating the other nodes, we hopefully don't disturb the other nodes, even while they might be locked. So there, the lock on the node is not absolute, we will change the record behind the back of another user holding a lock. On the other hand: messing up the sort order is less messy than messing with the actual content of a node. I'll take the risk. Worst case is that two processes will both update the sort order, perhaps yielding two nodes with the same sort_order value. Oh well, so be it. (There is this law by Pareto, something about 80 - 20. Mmmm...)

- **Uses \$DB**

void function PageManager::calc_home_id(\$node_id, 1) [*line 3981*]

Function Parameters:

- *int* **\$node_id** the node of interest
- *bool/int* **1** FALSE if no default node found, the default node_id otherwise

calculate the current default node on this level

this tries to find a sibling of the node \$node_id that has the flag 'is_default' set to TRUE

bool function PageManager::delete_node(\$node_id) [*line 1775*]

Function Parameters:

- *int* **\$node_id** the page or the section to delete

workhorse routine for deleting a node, including children

This deletes the children (but not grandchildren) of a section and the section itself OR simply the node itself. See function for more on this design decision.

This routine actually deletes nodes from the database, but only if these nodes do not have children AND if the nodes are not readonly. Furthermore, just before the child nodes are deleted, a lock on that node is obtained. This makes sure that a node that is currently being edited by another user is not deleted under her nose. Also, we do not delete nodes that have children because that would yield orphan nodes.

Any problems with deleting children are reported in messages via \$this->output. If all children are deleted successfully, then \$node_id is deleted. Success of the whole operation is indicated by returning TRUE, otherwise FALSE.

array function PageManager::get_dialogdef_add_node(\$is_page) [*line 2646*]

Function Parameters:

- *bool* **\$is_page** TRUE if the dialog is for a new page, FALSE is for a new section

construct a dialog definition for adding a node (page or section)

the dialog for pages and sections are different in just a single field: the page has an extra module field.

Note that we set two default values: one for visibility and one for the default module id. For

now we set the initial visibility to 2 (hidden). The default module is 1, under the assumption that the first module in the system is the one used most: a plain page. I didn't consider it worthy enough to make this defaults configurable. However, the sort order in the `get_options_modules()` doesn't guarantee that the plain page module is the first in the list, so there.

- **TODO** should we make the defaults in this routine configurable? (I'm not convinced they should)

array function PageManager::get_dialogdef_edit_advanced_node(\$node_id, \$is_page, [\$viewonly = FALSE], \$area_id) [line 2825]

Function Parameters:

- *int* **\$area_id** the area in which the node lives
- *int* **\$node_id** the node that is to be edited
- *bool* **\$is_page** TRUE if the dialog is for a page, FALSE is for a section
- *bool* **\$viewonly** if TRUE, most fields are 'dimmed' (uneditable)

construct a dialog definition for editing advanced properties of a node (page or section)

this constructs a dialog to edit the advanced properties of a node. There is a slight difference between pages and sections: a section can have neither the 'target' property nor the 'href' property; that only makes sense for a page, so these input fields are not displayed for a section.

The readonly-property is a special case. Even if the parameter `$viewonly` is TRUE, the readonly-field is displayed as 'editable'. This is because this particular field is used to toggle the viewonly mode: if a node is readonly, it cannot be edited, except the removal of the readonly attribute.

array function PageManager::get_dialogdef_edit_node(\$node_id, \$is_page, [\$viewonly = FALSE]) [line 2739]

Function Parameters:

- *int* **\$node_id** the node that is to be edited
- *bool* **\$is_page** TRUE if the dialog is for a page, FALSE is for a section
- *bool* **\$viewonly** if TRUE, all fields are 'dimmed' (uneditable) and there is no [Save] button

construct a dialog definition for editing basic properties of an existing node (page or section)

the dialog for pages and sections is different in just a single field: the page has an extra module field.

Note that we return a keyed array using the name of the dialog field as a key. This makes it easier to reference an incoming field in the save routine.

void function PageManager::get_dialog_data_node(&\$dialogdef, \$node_id) [line 2958]

Function Parameters:

- *array &\$dialogdef* the dialog definition
- *int \$node_id* the node that needs to be edited

fill the node dialog with data from the database

this fills a node dialog with data from the database. The routine takes care of some data conversions, e.g. manipulating a boolean TRUE/FALSE so it fits in a checkbox type of widget, etc.

Note that the data is NOT specifically validated. This means that a dialog `_could_` contain invalid values even when the user doesn't change anything. Or, to put it a different way: if the database contains garbage, the garbage is simply presented to the user. If the user subsequently tries to save the "garbage" the validation will catch her.

This routine is able to fill the values for both the 'basic' and the 'advanced' dialogs.

void function PageManager::get_icon_delete(\$node_id) [line 2354]

Function Parameters:

- *int \$node_id* the node to delete

construct a clickable icon to delete this node (and underlying nodes too)

- **TODO** should we display trash can icons for sections with non-empty subsections? there

really is no point, because we eventually will not accept deletion of sections with grandchildren in task_node_delete. Hmmmmmm..... For now I just added the condition that access is denied when a section has grandchildren. Need to refine this, later. Also, how about readonly nodes? Surely those cannot be deleted... should it not show in the icon?

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_icon_edit(\$node_id) [line 2389]

Function Parameters:

- *int* **\$node_id** the node to edit

construct a clickable icon to edit this node

- **TODO** move permission check to a separate function permission_edit_node()
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_icon_home(\$node_id) [line 2300]

Function Parameters:

- *int* **\$node_id** the node of interest

construct a clickable icon to set the home page/section on this tree level

this constructs a clickable icon to change the default node on this level. it requires PERMISSION_NODE_EDIT_PAGE or PERMISSION_NODE_EDIT_SECTION for both the target default node AND the current default node (if any)

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_icon_invisibility(\$node_id) [line 2431]

Function Parameters:

- *int* **\$node_id** the node to edit

construct a clickable icon to edit the advanced properties of this node

This icon has another purpose besides creating a link to the advanced properties: it also indicates whether a node is 'invisible' or not. In this context 'invisible' means either

- the node is under embargo until some time in the future, OR
- the node is already expired some time in the past, OR
- the node is hidden (ie. page is not visible in navigation but otherwise available).

Depending on the visibility a different icon is displayed.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_icon_page_preview(\$node_id) [line 2497]

Function Parameters:

- *int* **\$node_id** the node to preview

construct a clickable icon to preview this node

this constructs an icon to preview the page. the user should have edit permissions OR edit content permissions, because you can see the page when you can edit it, so there's no point in preventing the preview in that case. See [task_page_preview\(\)](#) for more information.

The preview is displayed in a separate window, either generated via a small routing in

javascript or (if javascript disabled) via a target="_blank".

- **TODO** if this is a public area, the user can see every page, except the expired/embargo'ed ones should we take that into account too? I'd say that is way over the top. How about pages in an intranet where the user has view privilege? Complicated. KISS: only show preview to those that can edit or edit content.
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_icon_section(\$node_id) [line 2545]

Function Parameters:

- *int* **\$node_id** the section to open/close

construct a clickable icon to open/close this node

This is a toggle: if the node is closed the closed icon is shown, but the action in the A-tag is to open the icon (and vice versa).

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::get_link_node_edit(\$node_id) [line 2589]

Function Parameters:

- *int* **\$node_id** the node for which to make the link

construct a clickable link to edit this node showing the page's title or link-text

this generates an A tag which leads to editing the properties (node == section) or content (node == page). Additional information displayed via the title attribute includes the node_id.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

array/bool function PageManager::get_module_records() [*line 3699*]

retrieve a list of all available module records

this returns a list of active module-records or FALSE if none are available The list is cached via a static variable so we don't have to go to the database more than once for this. Note that the returned array is keyed with module_id.

array function PageManager::get_node_id_and_ancestors(\$node_id) [*line 3752*]

Function Parameters:

- *int* **\$node_id** start at the youngest in the family

get an array with all ids of ancestors of node_id and node_id itself

note that the order of nodes is from top to bottom

array function PageManager::get_options_area(\$node_id, \$is_page) [*line 3186*]

Function Parameters:

- *int* **\$node_id** the node for which we are building this picklist
- *bool* **\$is_page** TRUE if this concerns a page, FALSE for a section

generate a list of areas for use in a dropdown list (for moving a node to another area)

this creates an array containing a list of areas to which the user is allowed to move a node. Permissions for moving a node is a combination of permissions for deleting a node from the current area, and adding a node to the target area. The current area \$this->area_id is always in the list, because even if the user isn't allowed to move a node to

somewhere else, she is at least allowed to leave the node in the area it currently is in. Therefore the option for the current area **MUST** be possible.

We sepcifically check for these permissions: `PERMISSION_AREA_ADD_PAGE` or `PERMISSION_AREA_ADD_SECTION` and not `PERMISSION_NODE_ADD_PAGE` or `PERMISSION_NODE_ADD_SECTION` because the target of the move is always the top level, and not some (sub)section.

array function PageManager::get_options_modules() [line 3221]

fetch a list of available modules for inclusion on a page

this retrieves a list of modules that can be used as a list of options in a listbox or radiobuttons. Only the active modules are considered. The names of the modules that are displayed in the list are translated (retrieved from the modules language files). The list is ordered by that translated module name.

array function PageManager::get_options_parents(\$is_page, [\$forbidden_id = NULL]) [line 3036]

Function Parameters:

- *bool \$is_page* if TRUE check page permissions, else check section permissions
- *mixed \$forbidden_id* identifies the subtree to EXclude from the results or NULL for all sections

construct an options list of possible parent sections

this constructs an array suitable for a radio field or a listbox. If the user has the privilege, an option 'add to toplevel' is added too.

If `$forbidden_id` is not NULL, it identifies the subtree that should be excluded from the result. If it were not excluded, the user might choose a child section as the parent for a section, which would introduce endless loops or circular references. Excluding the 'own' subtree prevents that.

Note that the list is constructed using recursion: the actual work is is done in the routine [get_options_parents_walk\(\)](#).

Also note that if `$forbidden_id` is not NULL, we interpret this as a request to generate a picklist of parents for that node. We make sure that we always add the current parent node to the list. This way the only option for a parent might be to keep the current one, which obviously should be one of the options.

- Uses [PageManager::get_options_parents_walk\(\)](#)

void function PageManager::get_options_parents_walk(&\$options, \$is_page, \$node_id, \$forbidden_id) [line 3082]

Function Parameters:

- *array* **&\$options** resulting array, output of this routine
- *bool* **\$is_page** distinction between page (TRUE) or section (FALSE)
- *int* **\$node_id** the subtree where we should start
- *int|null* **\$forbidden_id** if not NULL the subtree to skip

workhorse for construction an options list of possible parent sections

This routine is called recursively in order to construct a list of possible parent sections in the same order as the main tree display (see [show_tree\(\)](#)), but excluding the subtree starting at \$forbidden_id.

The list of parents is collected in \$options. This variable is passed by reference to save memory and also to keep the parents in the correct order.

Note that the options in the output array all have a parameter 'class' which can be used to detect how deep the nesting is. This can be visualised via wellchosen CSS-parameters, eg.

```
option.level0 { margin-left: 0px; }
option.level1 { margin-left: 20px; }
option.level2 { margin-left: 40px; }
...
```

which provides the illusion of a tree-like structure, even in a listbox.

The current parent of node \$forbidden_id is always included in the list of allowable parents because a node should be able to keep the current parent, always.

- Usedby [PageManager::get_options_parents_walk\(\)](#)
- Usedby [PageManager::get_options_parents\(\)](#)
- Uses [PageManager::get_options_parents_walk\(\)](#)

array function PageManager::get_options_sort_order(\$node_id) [line 3133]

Function Parameters:

- *int* **\$node_id** the node for which the list of siblings must be constructed

generate a list of siblings in a particular (sub)section used to select/change sort order via a list box

this constructs an (ordered) list of siblings of \$node_id, but excluding \$node_id itself. Also, an option 'sort at the top of the list' is included. This allows for selecting a sibling AFTER which \$node_id should appear in the section. The special value for 'before all others' or 'at the top of the list' is 0, because that value cannot be used by a real node.

- **Uses \$USER**
- **Uses \$CFG**

bool function PageManager::lock_records_subtree(\$node_id, \$new_area_id) [*line 3402*]

Function Parameters:

- *int* **\$node_id** the node which we are going to move to \$new_area_id
- *int* **\$new_area_id** the area to which we want to move the subtree \$node_id

attempt to lock all node records in a subtree

this recursively walks the subtree starting at \$node_id and attempts to

- lock every node in the subtree, AND
- write the new area_id in an auxiliary field of every node in the subtree

With at least two trips to the database (at least one for the lock and another one for writing the auxiliary field) per node, this is an expensive routine. Maybe it is possible to combine locking and writing the auxiliary field. However, in order to keep things readable I decided against that.

This routine returns FALSE if any of the nodes in the subtree could NOT be locked. If each and every node in the subtree is successfully locked, TRUE is returned.

Note that all these locks are reset/released the moment the actual move is done, by resetting both the locked_by field and the area_id field. That may hurt readability too, but less than combining lock + setting auxiliary field. See [save_node_new_area_mass_move\(\)](#) for more information.

- Usedby [PageManager::lock_records_subtree\(\)](#)
- Usedby [PageManager::save_node_new_area_mass_move\(\)](#)
- Uses [PageManager::lock_records_subtree\(\)](#)

string function PageManager::message_from_lockinfo(\$lockinfo, \$node_id, \$is_page) [*line 3783*]

Function Parameters:

- *array* **\$lockinfo** contains information about another user that has obtained a record lock
- *int* **\$node_id** the node that is locked
- **\$is_page**

construct a readable message from the lockinfo array

if an attempt to lock a record fails (see [lock_record_node\(\)](#)), the array \$lockinfo is filled with information about the user that has locked the record. The following information is available:

- 'user_id': the numerical user_id of the user holding the lock
- 'username': the userid of that user
- 'full_name': the full name of that user
- 'user_information': the IP-address from where that user is calling
- 'ctime': the date/time that user logged in (c=create)
- 'atime': the date/time that user last accessed the system (a=access)
- 'ltime': the date/time that user actually locked the record (l=lock)

This routine tries to construct a more or less readable message which informs this user here about that other user holding the lock.

bool function PageManager::module_connect(\$area_id, \$node_id, \$module_id) [*line 4076*]

Function Parameters:

- *int* **\$area_id** the area where \$node_id resides
- *int* **\$node_id** the node to which the module will be connected
- *int* **\$module_id** the module that will be connected to node \$node_id

inform module \$module_id that from now on it will be linked to page \$node_id

this routine tells module \$module_id that from now on it is associated with node \$node_id in area \$area_id.

This is done by a. loading the module's administrative interface (the admin-script file), and b. calling the function <modulename>_connect()

If something goes wrong (e.g. no module found, non-existing admin-script, undefined function <modulename>_connect()) FALSE is returned, otherwise the return value of function <modulename>_connect() is returned.

- **TODO** should we pass the area_id at all? What happens when a node is moved to another area without informing the module? Questions, questions, questions...

bool function PageManager::module_disconnect(\$area_id, \$node_id, \$module_id) [line 4038]

Function Parameters:

- *int \$area_id* the area where \$node_id resides
- *int \$node_id* the node from which the module is disconnected
- *int \$module_id* the module that will be disconnected from node \$node_id

inform module \$module_id that it is no longer linked to page \$node_id

this routine tells module \$module_id that it is no longer associated with node \$node_id in area \$area_id.

This is done by a. loading the module's administrative interface (the admin-script file), and b. calling the function <modulename>_disconnect()

If something goes wrong (e.g. no permissions, no module found, non-existing admin-script, undefined function <modulename>_disconnect()) FALSE is returned, otherwise the return value of function <modulename>_disconnect() is returned.

- **TODO** should we pass the area_id at all? What happens when a node is moved to another area without informing the module? Questions, questions, questions...

bool/array function PageManager::module_load_admin(\$module_id) [*line 4189*]

Function Parameters:

- *int* **\$module_id** indicates which module to load

load the admin interface of a module in core

this includes the 'admin'-part of a module via 'require_once()'. This routine first figures out if the admin-script file actually exists before the file is included. Also, we look at a very specific location, namely:
/program/modules/<modulename>/<module_admin_script> where
<modulename> is retrieved from the modules table in the database.

Note that if modulename would somehow be something like
"./.././.././.././etc/passwd\x00", we could be in trouble...

- **TODO** should we sanitise the modulename here? It is not user input, but it comes from the modules table in the database. However, if a module name would contain sequences of
"./../" we might be in trouble

bool function PageManager::module_save(\$node_id, \$module_id, \$viewonly, &\$edit_again) [*line 4154*]

Function Parameters:

- *int* **\$node_id** the node to which the module is connected
- *int* **\$module_id** the module that is connected to node \$node_id
- *bool* **\$viewonly** if TRUE, editing and thus saving is not allowed
- *bool* **&\$edit_again** returns TRUE if more editing is required, FALSE otherwise

(maybe) save the modified content of module \$module_id connected to page \$node_id

this saves the module data belonging to node \$node_id.

If something goes wrong (e.g. no module found, non-existing admin-script, undefined function <modulename>_save()) FALSE is returned, otherwise the return value of function <modulename>_save() is returned.

bool function PageManager::module_show_edit(\$node_id, \$module_id, \$viewonly, \$edit_again) [*line 4120*]

Function Parameters:

- *int* **\$node_id** the node to which the module is connected
- *int* **\$module_id** the module that is connected to node \$node_id
- *bool* **\$viewonly** if TRUE, editing is not allowed (but simply showing the content is allowed)
- *bool* **\$edit_again** if TRUE, start with data from \$_POST, otherwise read from database

show a dialog for editing the content of module \$module_id linked to page \$node_id

this loads the code for module \$module_id and calls the appropriate routine for displaying a dialog

The parameter \$viewonly can be used to indicate readonly access to the content. It is upto the called function to adhere to this flag, e.g. by just showing the content instead of letting the user modify it.

If the flag \$edit_again is TRUE, this is not the first call to this routine, i.e. we have been here before but probably something went wrong when saving the data (e.g. an invalid date like 2008-02-31 was entered). This makes it possible to re-edit the content without starting from scratch again. If the flag is FALSE, the called routine is supposed to start with the data as it is currently stored in the database. Otherwise the current data is POST'ed by the user.

If something goes wrong (e.g. no module found, non-existing admin-script, undefined function <modulename>_show_edit()) FALSE is returned, otherwise the return value of function <modulename>_show_edit() is returned.

- **Usedby** [PageManager::task_node_edit_content\(\)](#)

string function PageManager::node_full_name(\$node_id) [*line 3717*]

Function Parameters:

- *int* **\$node_id** get the full name of this node

shorthand for constructing a readable page/section name with id, name and title

bool function PageManager::node_has_grandchildren(\$node_id) [line 3729]

Function Parameters:

- *int* **\$node_id** the section to check for grandchildren

shorthand to determine whether the number of levels below section \$node_id is greater than one

bool function PageManager::permission_add_any_node(\$is_page) [line 3464]

Function Parameters:

- *bool* **\$is_page** selects either page or section permissions

does the user have the privilege to add a node, any node to an area?

this routine returns TRUE if the current user has permission to add at least one node to the current area. This information is used to show or suppress the 'add a page' and 'add a section' links.

Note that pages and sections are treated separately; if a user is allowed to add a page it doesn't necessarily mean that she is allowed to add a section too.

Strategy: we first check the area-level (and implicit site-level) permissions to add a node, anywhere in an area including at the toplevel. If that doesn't work, we check for permissions to add a node to an existing section at the node level (and implicit at the area and site level too). If that doesn't work, we return FALSE.

Note that it is enough to stop the search at the first hit: we need only 1 hit for 'any', not all of them.

- **Uses \$USER**

bool function PageManager::permission_add_node(\$section_id, \$is_page) [line 3496]

Function Parameters:

- *int* **\$section_id** is the section to examine or 0 for the area top level
- *bool* **\$is_page** selects either page or section permissions

does the user have the privilege to add a node the area or a section?

this checks for permission to add a page or a section to the area at the toplevel or to the section \$node_id. If access is denied initially, the upper sections are tested for the requested permission. I dubbed this cascading permissions (if a section allows for adding a page, any subsections inherit that permission). This routine is protected from endless loops by recursing at most MAXIMUM_ITERATIONS levels.

- **Uses \$USER**

bool function PageManager::permission_delete_node(\$node_id, \$is_page) [*line 3635*]

Function Parameters:

- *int* **\$node_id** is the node to delete
- **\$is_page**

does the user have the privilege to delete a node from the area?

- **TODO** we should also take the readonly flag into account (or should we?) when determining delete permissions
- **Uses \$USER**

bool function PageManager::permission_edit_node(\$node_id, \$is_page, [\$check_content = FALSE]) [*line 3561*]

Function Parameters:

- *int* **\$node_id** is the node to examine
- *bool* **\$is_page** TRUE means we look at page permissions, not section

- *bool* **\$check_content** TRUE means check edit content, else edit plain

does the user have the privilege to edit node properties?

this checks the edit permissions for the specified node. If none are found initially, we check out the permissions of the parent section. If the user allowed to add a page/section in the parent section, we assume or imply that the user also has edit permissions if she also has edit permissions for the parent. even though the exact permission bits are not set for this particular new node. IOW: if a user is able to add a page/section it would be illogical not to be able to edit the new page/section. However, if the edit-permissions are not area-wide, there is no way you can add permissions to a particular node before it exists. (Can't do that in the account manager).

Note that a node can also have the readonly attribute set. This is more or less a tool to prevent accidental changes to a node's properties: a user can easy reset the readonly flag and change the node anyway. However, it requires two steps and hence at least *_some_* thinking. Bottom line: we only look at the 'real' permissions here, and not the readonly flag. (Even better: edit privilege is required to reset the readonly flag so using that flag as extra permission would yield pages completely uneditable).

This routine is also used to check for content edit permissions. This is only possible for pages (not sections). By default this routine checks the regular permissions (edit properties/ edit advanced properties).

- **Uses \$USER**

bool function PageManager::permission_edit_node_content(\$node_id) [*line 3599*]

Function Parameters:

- *int* **\$node_id** is the node to examine

does the user have the privilege to edit node content?

this is a wrapper around routine [permission_edit_node\(\)](#). We force *is_page* and *check_content* to TRUE.

bool function PageManager::permission_set_default(\$node_id) [*line 3614*]

Function Parameters:

- *bool* **\$node_id** is the tentative new default

does the user have the privilege to make node **\$node_id** the default?

if a user has edit permission for the new default node and also in the existing default node (if any), the user is allowed to set the default to node **\$node_id**. Note that once again we use cascading permissions. (See also [permission_edit_node\(\)](#)).

void function PageManager::queue_area_node_alert(\$areas, \$nodes, \$alert_message, [\$username = "], \$message) [line 3309]

Function Parameters:

- *mixed* **\$areas** an array or a single int identifying the area(s) of interest
- *mixed* **\$nodes** an array or a single int identifying the node(s) of interest
- *string* **\$message** the message to add to the buffer of qualifying alert accounts
- *string* **\$username** (optional) the name of the user that initiated the action
- **\$alert_message**

add a message to message queue of 0 or more alerts

this adds **\$alert_message** to the message buffers of 0 or more alert accounts. The alerts that qualify to receive this addition via the `alerts_areas_nodes` table. The logic in that table is as follows:

- the `area_id` must match the `area_id(s)` (specified in **\$areas**) OR it must be 0 which acts as a wildcard for ALL areas
- the `node_id` must match the `node_id(s)` (specified in **\$nodes**) OR it must be 0 which acts as a wildcard for ALL nodes

Also the account must be active and the flag for the area/node-combination must be TRUE.

As a rule this routine is called with a single `area_id` in **\$areas** and a collection of `node_id`'s in **\$nodes**. The nodes follow the path up through the tree, in order to alert accounts that are only watching a section at a higher level.

Example: If user 'webmaster' adds new page, say 34, to subsection 8 in section 4 in area 1, you get something like this:

```
queue_area_node_alert(1,array(8,4,34),'node 34 added','webmaster');
```

The effect will be that all accounts with the following combinations of area A and node N have the message added to their buffers: A=0, N=1 - qualifies for all nodes in all areas A=1, N=0 -

qualifies for all nodes in area 1 A=1, N=4 - qualifies for node 4 in area 1 A=1, N=8 - qualifies for node 8 in area 1

It is very well possible that no message is added at all if there is no alert account watching the specified area and node (using wildcards or otherwise).

Near the end of this routine, we check the queue with pending messages, and perhaps send out a few alerts. The number of messages that can be sent from here is limited; we don't want to steal too much time from an unsuspecting user. It is the task of cron.php to take care of eventually sending the queued messages. However, sending only a few messages won't be noticed. I hope.

Note that this routine adds a timestamp to the message and, if it is specified, the name of the user.

Also note that the messages are added to the buffer with the last message at the top, it means that the receiver will travel back in time reading the collection of messages. This is based on the assumption that the latest messages sometimes override a previous message and therefore should be read first.

- **Uses \$DB;**

void function PageManager::save_node(\$node_id) [line 1900]

Function Parameters:

- *int \$node_id* the node we have to change

workhorse routing for saving modified node data to the database

this is the 'meat' in saving the modified node data. There are a lot of complicated things we need to take care of, including dealing with the readonly property (if a node is currently readonly, nothing should be changed whatsoever, except removing the readonly attribute) and with moving a non-empty section to another area. Especially the latter is not trivial to do, therefore it is being done in a separate routine (see [save_node_new_area_mass_move\(\)](#)).

Note that we need to return the user to the edit dialog if the data entered is somehow incorrect. If everything is OK, we simply display the treeview and the area menu, as usual.

Another complication is dealing with a changed module. If the user decides to change the module, we need to inform the old module that it is no longer connected to this page and is effectively 'deleted'. Subsequently we have to tell the new module that it is in fact now added

to this node. It is up to the module's code to deal with these removals and additions (for some modules it could boil down to a no-op).

Finally there is a complication with parent nodes and sort order. The sort order is specified by the user via selecting the node AFTER which this node should be positioned. However, this list of nodes is created based on the OLD parent of the node. If the node is moved to elsewhere in the tree, sorting after a node in another branch no longer makes sense. Therefore, if both the parent and the sort order are changed, the parent prevails (and the sort order information is discarded).

- **TODO** this routine could be improved by refactoring it; it is too long!
- **TODO** there is something wrong with embargo: should we check starting at parent or at node? this is not clear: it depends on basic/advanced and whether the embargo field changed. mmmm... safe choice: start at node_id for the time being

bool function PageManager::save_node_new_area_mass_move(\$node_id, \$new_area_id, \$embargo) [line 2190]
Function Parameters:

- *int \$node_id* the node which we are going to move to \$new_area_id
- *int \$new_area_id* the area to which we want to move the subtree \$node_id
- *bool \$embargo* if TRUE, we cannot send alerts because the original tree is under embargo

workhorse routine for moving a complete subtree to another area

this routine moves a subtree starting at section \$node_id from area \$this->area_id to area \$new_area_id. This is a complicated operation because

- the subtree may be (very) large
- nodes in the subtree may be locked by other users
- we MUST take an all or nothing approach because either ALL of the nodes or NONE of the nodes

in the subtree must change the area_id. If area_id's are only changed partly, we will end up

with orphan nodes because areas differ between a parent and corresponding offspring.

Here is my train of thoughts leading to my implementation.

The best solution I can think of is to:

- lock all individual nodes in the subtree, and if successful,
- update all these records with the appropriate area_id and at the same time unlocking these records by writing a NULL to the lock field.

This way we postpone the actual move to the new area until we are certain that we have all nodes involved in our hands. If we don't succeed in obtaining all the necessary locks, we have to abandon the operation, accept defeat, release all locks and return FALSE to our caller. If we do succeed, well, we indicate success by returning TRUE.

Mmmm....

Note that the user might have two browser windows open in the same session. This shouldn't happen, but there is no easy way to prevent the user to open more windows in the same session. This may lead to an undesirable result: if the user is editing another node in the same session in another window (totally unrelated to the move of the current subtree), that node might also be moved to the new area, introducing an orphan in the new area. Mmmmm. The best way to handle that problem is to use a special helper field, say `auxiliary_id`, in the nodes table. That field could be used as follows (pseudo-code):

```
set auxiliary_id of $node_id to $new_area_id for all descendants of section $node_id do
obtain lock on descendant    set auxiliary_id to $new_area_id update nodes set area_id =
new_area_id, auxiliary_id = NULL, locked_by = NULL where auxiliary_id = new_area_id AND
locked_by = our_session_id Then we once again concentrate the actual work in a single
UPDATE-statement.
```

It is a costly operation: at least 2 trips to the database per descendant.

Mmmmm...

Perhaps we can save (a lot) of trips to the database if we build on the assumption that usually there are more children in every section AND that usually the children are NOT locked. In that case the pseudo-code becomes:

```
for all descendants of section $node_id do    if is_section($descendant) then        SET
auxiliary_id = $new_area_id, locked_by = $our_session_id WHERE                locked_by IS
NULL AND parent_id = $descendant;    endif endfor SET area_id = new_area_id, auxiliary_id
= NULL, locked_by = NULL    WHERE auxiliary_id = new_area_id AND locked_by =
$our_session_id
```

However, we might miss a descendant or two if it happens to be locked (by us, or by another session). That's no good.

Mmmmm...

I'm sure there's a better way, but for the time being I'll simply use brute force and my way through the subtree. If this really becomes a huge problem, we may want to refactor this routine.

- Uses [PageManager::lock_records_subtree\(\)](#)

bool function PageManager::section_is_open(\$section_id) [*line 3675*]

Function Parameters:

- *int* **\$section_id** the section of interest

shorthand for determining whether a section is opened or closed

void function PageManager::show_area_menu([\$current_area_id = NULL]) [*line 1311*]

Function Parameters:

- *int|null* **\$current_area_id** the current area

construct a clickable list of available areas for the current user

this iterates through all available areas in the areas table, and constructs a list of areas (as LI's in a UL) for which the current user has either administrative or view permissions. The latter shows in 'dimmed' form, because it is not allowed to view this area in pagemanager, but the area does exist and is available to the user (as a visitor rather than an administrator) so it should not be suppressed. If a user has neither view or admin permission, the area is suppressed. Note that every user has at least view permissions for a public area.

The current area is determined by parameter `$current_area_id`. This area gets the attribute 'class="current"' which makes it possible to emphasise the current working area in the menu (via CSS).

void function PageManager::show_dialog_delete_node_confirm(\$node_id) [*line 1715*]

Function Parameters:

- *int* **\$node_id** the page or the section to delete

display a list of 1 or more nodes to delete and ask user for confirmation of delete

this displays a confirmation question with a list of nodes that will be deleted. This list is either a single page or a single (empty) section OR a section with children (but not grandchildren). See function [task_node_delete\(\)](#) for more on this design decision. If the user presses Delete button, the nodes will be deleted, if the user presses Cancel then nothing is deleted.

`void function PageManager::show_edit_menu($node_id, [$is_page = FALSE], [$current_option = NULL])` [line 1366]
Function Parameters:

- *int* **\$node_id** the current node (the node being edited)
- *bool* **\$is_page** if TRUE display the link to edit content too (this is for pages only)
- *int* **\$current_option** the currently selected edit mode (basic, advanced or content)

construct a clickable list of edit variants (basic, advanced and maybe content)

this constructs a menu from where the user can navigate to edit basic properties of a node, advanced properties or even the content (for pages).

- **Uses** `$WAS_SCRIPT_NAME`
- **Uses** `$USER`
- **Uses** `$CFG`

`void function PageManager::show_tree()` [line 1442]

create a tree-like list of nodes in the content area of \$this->output

this constructs a tree-like view of the current area, with

- a title
- 0, 1 or 2 links to add a node
- 0, 1 or 2 links to select a different tree view
- all nodes that are currently show-able (depending on tree view mode)

If the tree is empty, only the links to add a node are displayed (if the user has permission to add). The individual nodes are displayed using recursion with [show_tree_walk\(\)](#).

Note that the tree is constructed via nested UL's with LI's, all in name of 'graceful degradation': this interface still works if this program has no stylesheet whatsoever).

- **Uses** [PageManager::show_tree_walk\(\)](#)
- **Uses** `$WAS_SCRIPT_NAME`

- **Uses \$USER**
- **Uses \$CFG**

`void function PageManager::show_treeview_buttons()` [line 1632]

show one or two clickable links to change the view of the tree

There are three different tree views:

- **minimal:** all sections are closed, only the top level nodes are shown
- **custom:** 1 or more sections are closed and 1 or more sections are opened
- **maximal:** all sections are opened, all nodes are shown

There is a fourth option:

- **none:** there are no sections at all

The view can be set to either `TREE_VIEW_MINIMAL`, `TREE_VIEW_CUSTOM` or `TREE_VIEW_MAXIMAL`. The current setting is remembered in session variable 'tree_mode'. A list of customised nodes is kept in session variable `expanded_nodes[]`, an array keyed with the node number and a value of either `TRUE` (section is 'open') or `FALSE` (section is 'closed'). An empty array implies all nodes are closed (ie. default value is `FALSE`).

In some cases `TREE_VIEW_CUSTOM` is equivalent to one of the other two, e.g. when the user closes the last section, the effect looks exactly like `TREE_VIEW_MINIMAL`. If the user manually opens all sections, the effect is the same as `TREE_VIEW_MAXIMAL`.

In this routine we want to show 0, 1 or 2 buttons that allow the user to switch to another viewmode, but only if the new mode(s) are different from the current one.

The equivalency between modes can be determined by counting the number of open and closed sections. Here is a truth table.

N	open	closed	description
0	0	0	no sections at all, show 0 buttons (all modes are equivalent)
1	0	>=1	all sections are closed, 'custom' is equivalent with 'minimal'
2	>=1	0	all sections are opened, 'custom' is equivalent with 'maximal'
3	>=1	>=1	some open, some closed, 'custom' is distinct from the other two modes

Case N=0 In this case there are no sections at all, so there is no point to show any button at all because all views are equivalent: all available pages (if any) live at the top level and they are always visible.

Case N=1 In this case 'minimal' and 'custom' are equivalent. That means that if the current view is either 'minimal' or 'custom', the only viable option would be to set the view to 'maximal'. If the current mode is 'maximal', the only viable option is 'minimal'. Only 1 toggle-like button needs to be displayed.

Case N=2 In this case 'custom' and 'maximal' are equivalent. That means that if the current view is either 'custom' or 'maximal', the only viable option would be to set the view to 'minimal'. If the current mode is 'minimal', the only viable option is 'maximal'. Only 1 toggle-

like button needs to be displayed.

Case N=3 In this case 'custom' is a distinct mode somewhere between 'minimal' and 'maximal'. This means that there are always two other options to choose from: if current mode is 'minimal' the choices are 'custom' and 'maximal', if current mode is 'custom' the choices are 'maximal' and 'minimal', if current mode is 'maximal' the choices are 'minimal' and 'custom'. This means that two buttons need to be displayed.

Strategy: First we step through the tree and we count the 'open' and 'closed' sections. After that we determine whether N is 0,1,2 or 3 (see truthtable). After that we calculate which of the three buttons need to be displayed, depending on the current mode (obtained via the session variable 'tree_mode'). Subsequently the buttons are output to the 'content' area via \$this->output.

- **Uses** \$WAS_SCRIPT_NAME

void function PageManager::show_tree_walk(\$node_id, [\$m = ""]) [*line 1517*]

Function Parameters:

- *int* **\$node_id** the first node of this tree level to show
- *string* **\$m** left margin for increased readability

display the specified node, optionally all subtrees, and subsequently all siblings

this routine displays the specified node, including clickable icons for setting the default, editing the node etc. from the current tree. After that, any subtrees of this node are displayed using recursion (but only if the section is 'opened'). This continues for all siblings of the specified node until there are no more (indicated by a sibling_id equal to zero).

- **Usedby** [PageManager::show_tree\(\)](#)
- **Usedby** [PageManager::show_tree_walk\(\)](#)
- **Uses** [PageManager::show_tree_walk\(\)](#)
- **Uses** \$WAS_SCRIPT_NAME

- **Uses \$USER**
- **Uses \$CFG**

void function PageManager::task_node_add(\$task) [line 516]

Function Parameters:

- *string \$task* identifies whether a page or a section should be added

display a dialog to add a new page or section to the current area

this displays a dialog where the user can add a node to the current area. If the user has no permissions to add a node at all, the result is an error message and the tree view

The value of \$task (which can be either TASK_ADD_PAGE or TASK_ADD_SECTION) determines which dialog to show.

Both dialogs are very similar (a page can have a module, a section cannot). The actual dialog is constructed based on a dialogdef, see the function [get_dialogdef_add_node\(\)](#).

- **Uses \$USER**

void function PageManager::task_node_delete() [line 592]

delete one or more nodes from an area after user confirmation

this deals with deleting nodes from an area. There are two stages. Stage 1 is presenting the user with a list of selected nodes and offering the user the choice to confirm the delete or cancel the operation. Stage 2 is actually deleting the selected nodes (after the user confirmed the delete in stage 1), including the disconnection of pages and modules.

An important design decision was to limit the delete process to at most 1 tree level. This means the following. If a user attempts to delete a page, it is easy: after confirmation a single node is deleted from the database. If a user attempts to delete a section, it can be different. If the section is empty, i.e. there are no children, it is the same as deleting a page: only a single node record has to be deleted.

It becomes more dangerous if a section is filled, ie. has children. If all children are pages (or empty subsections), it is still relatively innocent because the worst case is that all pages in a section are deleted. If, however, the section contains subsections which in turn contain subsections, etc. the delete operation may become a little too powerful. If it would work

that way (deleting a section implies `_all_` nodes in the subtree), it is possible to delete a complete area in only a few keystrokes, no matter how many levels.

In order to prevent this mass deletion, we decided to limit the delete operation to at most a single level. In other words: the user can delete

- a single page
- a single empty section
- a section with children but no grandchildren

If a user attempts to delete a section with children and grandchildren, an error message is displayed and nothing is deleted.

This forces the user to delete a complete tree a section at the time, hopefully preventing a 'oh no! what have I done' user experience.

We `_always_` want the user to confirm the deletion of a node, even if it is just a single page.

Note that a page that is readonly will not be deleted.

- **TODO** should we display trash can icons for sections with non-empty subsections in treeview? there really is no point, because we eventually will not accept deletion of sections with grandchildren. Hmmmmm.....
- **Uses** \$USER

`void function PageManager::task_node_edit($task) [line 677]`

Function Parameters:

- *string* **\$task** identifies whether the basic or advanced properties should be edited

display a dialog where the user can edit basic or advanced properties of a node

this constructs a dialog and a menu where the user can edit the properties of a node. We check the user's permissions and if that works out we try to obtain a lock on the record. If that succeeds, we show the dialog (in funnel mode). If we don't get the lock, we inform the user about the other user who holds the lock. In case of error (e.g. no permissions or no lock) we fall back on displaying the area menu and the treeview.

Note: the lock is released once the user saves the node OR cancels the edit operation.

- **Uses** \$USER

void function PageManager::task_node_edit_content() [line 753]

display a dialog where the user can edit the contents of a node via a module

this effectively loads the module code associated with the specified node and subsequently calls the corresponding code in the module to display an edit dialog.

Just like the other edit routine (see [task_node_edit\(\)](#)) the node is locked first. Also the user permissions are checked. If we don't get the lock, we inform the user about the other user who holds the lock. In case of error (e.g. no permissions or no lock or an error loading the module) we fall back on displaying the area menu and the treeview. In that process the lock may be released.

- **Uses** [PageManager::module_show_edit\(\)](#)
- **Uses** \$USER

void function PageManager::task_page_preview() [line 882]

preview a page that is maybe still under embargo/already expired

if the user has permissions to preview the specified page, she is redirected to the regular site with a special one-time permission to view a page, even if that page is under embargo or already expired (which normally would prevent any user from viewing that page).

There are several ways to implement such a one-off permit, e.g. by setting a quasi-random string in the session and specifying that string as a parameter to index.php. If (in index.php) the string provided matches this string in the session, the user is granted access. However, this leaves room for the user to manually change the node id to `_any_` number, even a node that that user is not supposed to see.

Another solution might have been to simply include index.php. I decided against that; I don't want to have to deal with a mix of admin.php and index.php-code in the same run.

I took a slightly different approach, as follows. First I generate a quasi-random string of N (N=32) characters. (The length of 32 is an arbitrary choice.) This string is stored in the session variable. Then I store the requested node in the session variable, too. After that I calculate the md5sum of the combination of the random string and the node id. This yields a hash. This hash is passed on to index.php as the sole parameter.

Note that the quasi-random key never leaves the server: it is only stored in the session

variables. Also, the node id is not one of the parameters of index.php, this too is only stored in the session variables.

Once index.php is processed, the specified md5sum is retrieved and a check is performed on the node id and the quasi-random string in the session variables in order to see if the hashes match. If this is the case, index.php can proceed to show the page preview. Note that there is no way for the user to manipulate the node id, because that number never travels to the user's browser in plain text.

Making a bookmark for the preview will use the hash, but the hash depends on a quasi-random string stored in the session. It means that when the session is terminated, the bookmarked page will no longer be visible, which is good. Also, whenever another page preview is requested, a new quasi-random string is generated, which also invalidates the bookmarked page.

The only thing that CAN happen is that the user saves the preview in a place where it can be seen by others. Also, the page will probably be cached in the user's browser.

With respect to permissions: I consider the preview privilege equivalent with edit permission: if the user is able to edit the node she can see the content of the node anyway. However, maybe we should look at different permissions. Put it on the todo-list.

- **TODO** the check on permissions can be improved (is PERMISSION_XXXX_EDIT_NODE enough?)
- **TODO** there is an issue with redirecting to another site: officially the url should be fully qualified (ie. \$CFG->www). I use the shorthand, possibly without scheme and hostname (\$CFG->www_short). This might pose a problem with picky browsers. See [calculate uri shortcuts](#) for more information.
- **Uses** \$USER
- **Uses** \$CFG

void function PageManager::task_save_content() [line 1200]
void function PageManager::task_save_newnode(\$task) [line 986]

Function Parameters:

- *string* **\$task** distinguishes between saving a page or a section

save a newly added node to the database

this validate and save the (minimal) data for a new node (section or page). First we

check which button press brought us here; Cancel means we're done, else we need to validate the user input. This is done by setting up the same dialog structure as we did when presenting the user with a dialog in the first place. This ensures that WE determine which fields we need to look for in the `_POST` data. (If we simply were to look for fieldnames in the `_POST` array, we might be tricked in accepting random fieldnames. By starting from the dialog structure we make sure that we only look at fields that are part of the dialog; any other fields are ignored, minimising the risks of the user trying to trick us.)

The dialog structure is filled with the data POST'ed by the user and subsequently the data is validated against the rules in the dialog structure (eg. min length, min/max numerical values, etc). If one or more fields fail the tests, we redo the dialog, using the data from `_POST` as a new starting point. This makes that the user doesn't lose all other field contents if she makes a minor mistake in entering data for one field.

If all data from the dialog appears to be valid, it is copied to an array that will be used to actually insert a new record into the nodes table. This array also holds various other fields (not part of the dialog) with sensible default values. Interesting 'special' fields are 'sort_order' and 'is_hidden' and 'embargo'.

'sort_order' is calculated automatically from other sort orders in the same parent section. There are two ways to do it: always add a node at the end or the exact opposite: always add a node at the beginning. The jury is still out on which of the two is the best choice (see comments in the code below).

'is_hidden' and 'embargo' are calculated from the dialog field 'node_visibility'. The latter gives the user three options: 'visible', 'hidden' and 'embargo'. This translates to the following values for 'is_hidden' and 'embargo' (note that `$now` is the current time in the form 'yyyy-mm-dd hh:mm:ss'):

visible: `is_hidden = FALSE`, `'embargo' = $now`
hidden: `is_hidden = TRUE`, `'embargo' = $now`
embargo: `is_hidden = TRUE`, `'embargo' = '9999-12-31 23:59:59'`

This makes sure that IF the user wants to create a 'secret' node, ie. under embargo until some time in the future, the new node is never visible until the user edits the node to make it visible. However, there is no need to manually add a date/time: we simply plug in the maximum value for a date/time, which effectively means 'forever'.

Finally, if the new node is saved, a message about this event is recorded in the logfile (even for new nodes under embargo). Also, if the node is NOT under embargo, an alert message is queued. Note that we do NOT send alerts on a page that is created under embargo. (There is a slight problem with this: once a user edits the node and sets the embargo to a more realistic value, e.g. next week, there is no practical way to inform the 'alert-watchers' about that fact: we cannot send an alert at the time that the embargo date is changed to 'next week' because the node is still under embargo. We don't have a handy opportunity to send alerts because the embargo date will eventually come around and the node will become visible automatically, without anyone being alerted to the fact. Mmmm....

- **TODO** about 'sort_order': do we insert nodes at the end or the beginning of a parent section?
- **TODO** how do we alert users that an embargo date has come around? Do we schedule alerts via cron?

void function PageManager::task_save_node() [line 1111]

void function PageManager::task_set_default() [line 420]

make the selected node the default for this level

this sets a default node. First we make sure we have a valid environment and a node that belongs to the current area Then we check permissions and if the user is allowed to

- set the default bit on the target node, AND
- reset the default bit on the current default node
we actually
- reset the default bit from the current default (if there is one), AND
- set the default bit for the selected node.

Note: if the user sets the default node on the current default node, the default is reset and subsequently set again (two trips to the database), This also updates the mtime of the record.

- **Uses \$USER**

void function PageManager::task_subtree_collapse() [line 368]

close the selected section and perhaps change the view mode

this closes the selected node, i.e. fold in the subtree starting at the selected node. This should only happen when the view mode is either maximal (all sections closed) or custom (some sections opened and some sections closed). It should never happen when mode is minimal.

The status of a node (opened or closed) is remembered in session variable 'expanded_nodes': an array keyed with node_id If the corresponding value is TRUE, the section is considered open, all other values (FALSE or element is non-existing) equate to closed. See also [task_subtree_expand\(\)](#).

If the current mode is 'maximal', all sections are showed 'open'. When one of the sections is closed (via this routine), we change the mode to 'custom'. However, because the previous state was 'all sections are opened', we need to remember all the sections in the session variable 'expanded_nodes' and set them all to TRUE except the section that needs to be closed. We do this by constructing the complete tree of the area and adding an entry for every section and setting the value to TRUE, except the node that needs to be closed.

- **Uses \$USER**

void function PageManager::task_subtree_expand() [line 317]

open the selected section and perhaps change the view mode

this opens the selected node, i.e. unfold 1 level of the subtree starting at the selected node. This should only happen when the view mode is either minimal (all sections closed) or custom (some sections opened and some sections closed). It should never happen when mode is maximal.

The status of a node (opened or closed) is remembered in session variable 'expanded_nodes': an array keyed with node_id. If the corresponding value is TRUE, the section is considered open, all other values (FALSE or element is non-existing) equate to closed. See also [task_subtree_collapse\(\)](#).

- **Uses \$USER**

void function PageManager::task_treeview() [line 247]

maybe change the current area and then show the tree and the menu for the current area

this routine switches to a new area if one is specified and subsequently displays the tree of the new area or the existing current area.

void function PageManager::task_treeview_set() [line 283]

this sets the tree view to the specified mode

this is a simple routine to set the current view to one of the three possible views. The problem that sometimes 'custom' yields a view identical with 'maximal' or 'minimal' is dealt with when constructing the links to this routine [task_treeview_set\(\)](#). See [show_treeview_buttons\(\)](#) for more information.

Class Theme

[line 34]

Methods to access properties of a theme

- **Package** wascore

Theme::\$area_id

int = NULL [line 42]

- **Var** \$area_id the area to display

Theme::\$area_record

a = NULL [line 45]

- **Var** copy of the area record from the areas table

Theme::\$breadcrumb_addendum

array = array() [line 93]

- **Var** \$breadcrumb_addendum holds an array with additional anchors that can be set by the page's module

Theme::\$config

array = [line 57]

- **Var** \$config all properties from themes_areas_properties for this combination of theme and area

Theme::\$content

array = array() [line 72]

- **Var** collection of items/lines that are part of the content area

Theme::\$domain

string = [line 96]

- **Var** \$domain the language domain where we get our translations from, usually 't_<themenam>'

Theme::\$dtd

string = <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> [line 60]

- **Var** the standard doctype (default: HTML 4.01 Transitional)

Theme::\$friendly_url

bool = TRUE [line 90]

- **Var** \$friendly_url if TRUE, links via index.php/nnn/book_mark_friendly_text otherwise index.php?node=nnn

Theme::\$high_visibility

bool = FALSE [line 84]

- **Var** this switches the navigation between image-based and text-based

Theme::\$html_head

array = array() [line 69]

- **Var** collection of items/lines that will be output as part of the HTML-head section

Theme::\$http_headers

array = array() [line 66]

- **Var** collection of individual http-headers that are to be sent *_before_* any HTML is sent

Theme::\$messages_bottom

array = array() [line 81]

- **Var** collection of messages that are to be displayed via a javascript alert() at END of page

Theme::\$messages_inline

array = array() [line 78]

- **Var** collection of messages that are to be displayed inline, contained within the HTML body

Theme::\$messages_top

array = array() [line 75]

- **Var** collection of messages that are to be displayed via a javascript alert() at START of page

Theme::\$node_id

int = NULL [line 48]

- **Var** \$node_id the node (page) to display

Theme::\$node_record

a = NULL [line 51]

- **Var** convenient copy of the node record copied from the area tree

Theme::\$preview_mode

bool = FALSE [line 87]

- **Var** \$preview_mode if TRUE, we are previewing a page (from pagemanager)

Theme::\$theme_id

int = NULL [line 39]

- **Var** \$theme_id primary key of the theme

Theme::\$theme_record

a = NULL [line 36]

- **Var** copy of the corresponding record from the themes table

Theme::\$title

string = [line 63]

- **Var** the title to display in both the title tag and in the page itself (usually the areaname)

Theme::\$tree

array = FALSE [line 54]

- **Var** \$tree all nodes in area \$area_id, keyed by \$node_id (see [build_tree\(\)](#)).

Constructor *void* function Theme::Theme(\$theme_record, \$area_id, \$node_id) *[line 110]*

Function Parameters:

- *array* **\$theme_record** the record straight from the database
- *int* **\$area_id** the area of interest
- *int* **\$node_id** the node that will be displayed

construct a Theme object

this stores the information about this theme from the database. Also, we construct/read the tree of nodes for this area \$area_id. This information will be used later on when constructing the navigation. The node to display is \$node_id.

void function Theme::add_content(\$content) [line 885]

Function Parameters:

- *string/array* **\$content** the line(s) of text to add

add a line or array of lines to the content part of the document

void function Theme::add_html_header(\$headerline) [line 799]

Function Parameters:

- *string* **\$headerline** headerline to add

add a header to the HTML head part of the document

void function Theme::add_http_header(\$headerline) [line 790]

Function Parameters:

- *string* **\$headerline** headerline to add

add an HTTP-header

void function Theme::add_message(\$message) [line 837]

Function Parameters:

- *string/array* **\$message** message(s) to add inline

add a message to the list of inline messages, part of the BODY of the document

void function Theme::add_meta(\$meta) [line 862]

Function Parameters:

- *array* **\$meta** an array with name-value-pairs that should be added to the HTML head part

add a line with meta-information to the HTML head part of the document

void function Theme::add_meta_http_equiv(\$meta) [line 874]

Function Parameters:

- *array* **\$meta** an array with name-value-pairs that should be added to the HTML head part

add a line with http-equiv meta-information to the HTML head part of the document

void function Theme::add_popup_bottom(\$message) [line 823]

Function Parameters:

- *string|array* **\$message** message(s) to add

add a message to the list of popup-messages at the BOTTOM of the document

void function Theme::add_popup_top(\$message) [line 809]

Function Parameters:

- *string|array* **\$message** message(s) to add

add a message to the list of popup-messages at the TOP of the document

void function Theme::add_stylesheet(\$url) [line 851]

Function Parameters:

- *string* **\$url** url of the stylesheet

add a link to a stylesheet to the HTML head part of the document

array function Theme::calc_breadcrumb_trail(\$node_id) [line 1043]

Function Parameters:

- *int* **\$node_id** the node for which to calculate/set the path to the root node

set breadcrumbs in tree AND construct list of clickable anchors

Note: the anchors are created with the current setting of the preview mode, so if that changes after we construct a list of anchors we're in trouble. I prefer late binding, so the real list to use should be created in the phase where the HTML-code is constructed. Mmmmm...

- **TODO** split into two separate routines, one to set the tree, another to construct the list of anchors

void function Theme::calc_tree_visibility(\$node_id, &\$tree, [\$force_invisibility = FALSE]) [line 995]

Function Parameters:

- **\$node_id**
- **&\$tree**
- **\$force_invisibility**

calculate the visibility of the nodes in the tree

- **TODO** how about making all nodes under embargo visible when previewing a page or at least the path from the node to display?

void function Theme::construct_tree(\$area_id) [line 950]

Function Parameters:

- **\$area_id**

void function Theme::dump_subtree(\$node_id, &\$tree) [line 961]

Function Parameters:

- **\$node_id**
- **&\$tree**

string function Theme::friendly_bookmark(\$title) [*line 1137*]

Function Parameters:

- *string* **\$title** input text

construct an alphanumeric string from a node title (for a readable bookmark)

this strips everything from \$title except alphanumerics and spaces. The spaces are translated to an underscore. Length of result is limited to an arbitrary length of 50 characters.

string function Theme::get_address([\$m = ""]) [*line 768*]

Function Parameters:

- *string* **\$m** left margin for increased readability

return the reconstructed URL in a single (indented) line

This constructs the exact URL (including the GET-parameters) of the current script. This URL is returned as HTML so it can be displayed. It is NOT meant to be a clickable link, but as a documentation of the actual URL that was used. Note that this URL can be suppressed by an appropriate 'display:none' in the stylesheet, making it an item that only appears on a hardcopy (media="print") and not on screen.

- **TODO** should we add additional text to the address, e.g. prefix 'URL: ' or something? now it is just a plain old URL without any comments whatsoever.

string function Theme::get_bottomline([\$m = ""]) [*line 742*]

Function Parameters:

- *string* **\$m** left margin for increased readability

show 'powered by' and (maybe) report basic performance indicators

This calculates the execution time of the script and the number of queries. Note a special trick: we retrieve the translated string in a dummy variable before calculating the number of

queries because otherwise we might miss one or more query from the language/translation subsystem.

Note: for the time being the performance report commented out (2010-12-08).

string function Theme::get_content([\$m = ""]) [*line 349*]

Function Parameters:

- *string* **\$m** left margin for increased readability

get all lines in the content DIV in a single properly indented string

void function Theme::get_div_breadcrumbs([\$m = ""]) [*line 442*]

Function Parameters:

- **\$m**

construct breadcrumb trail

- **TODO** how about adding a title to the items? or do we do that already?

string function Theme::get_div_messages([\$m = ""]) [*line 388*]

Function Parameters:

- *string* **\$m** left margin for increased readability

get a perhaps bulleted list of messages in a DIV

This constructs an unordered list with messages, if there are any. If there is no message at all, an empty string is returned (without DIV). If there is a single message, no bullet is added to the message. If there are two or more messages, bullets are added.

Note that this routine is an exception with respect to the DIV-tags: this helper routine DOES generate its own DIVs whenever there is at least 1 message. This means that there is no DIV at all when there are no messages.

string function Theme::get_html() [*line 217*]

construct an output page in HTML

This constructs a full HTML-page, starting at the DTD and ending with the html closing tag.

The page is constructed using nested DIVs, the layout is taken care of in a separate style sheet. All knowledge about the structure of the page is contained in this routine.

The performance of the script (# of queries, execution time) is calculated as late as possible, to catch as much as we can. Therefore the construction is done in two parts and performance is calculated last.

The contents of the various DIVs is constructed in various helper routines in order to make this routine easy to read (by humans that is). The various helper routines all are called with a string of space characters; this should improve the the readability of the page that is generated eventually.

Note that the routine \$this->get_div_messages() does in fact generate its own DIV tags. This is done in order to completely get rid of the message DIV, we do not even want to see an empty DIV if there are no message.

The same logic applies to the breadcrumb trail.

string function Theme::get_html_head([\$m = ""]) [*line 288*]

Function Parameters:

- *string* **\$m** left margin for increased readability

get all lines in the HTML head section in a single properly indented string

- **TODO** also deal with Bazaar Style Style Sheets at node level in this routine

string function Theme::get_jumpmenu([\$m = ""]) [*line 684*]

Function Parameters:

- *string* **\$m** add readability to output

construct a simple jumplist to navigate to other areas

this constructs a listbox with areas to which the current user has access. The user can pick an area from the list and press the [Go] button to navigate to that area. Only the active areas are displayed. Private areas are only displayed when the user actually has access to those areas.

string function Theme::get_lines(\$a, [\$m = ""]) [line 362]

Function Parameters:

- *string* **\$m** left margin for increased readability
- **\$a**

get lines from an array in a single properly indented string

This is a workhorse to convert an array of lines to a properly indented block of text.

string function Theme::get_logo([\$m = ""]) [line 481]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct an image tag with the area logo

This constructs HTML-code that displays the logo.

- **TODO** should we take path_info into account here too???? how about /area/aaa/node/nnn instead of /aaa/nnn???

void function Theme::get_menu([\$m = ""]) [line 622]

Function Parameters:

- **\$m**

void function Theme::get_navigation([\$m = "], [\$textonly = FALSE]) [*line 602*]

Function Parameters:

- **\$m**
- **\$textonly**

string function Theme::get_popups(\$messages, [\$m = "]) [*line 423*]

Function Parameters:

- *string* **\$m** left margin for increased readability
- *array* **\$messages** @messages a collection of message to display via alert()

construct javascript alerts for messages

This constructs a piece of HTML that yields 0 or more calls to the javascript alert() function, once per message. If no messages need to be displayed an empty string is returned.

bool/array function Theme::get_properties(\$theme_id, \$area_id) [*line 941*]

Function Parameters:

- *int* **\$theme_id**
- *int* **\$area_id**

retrieve configuration parameters for this combination of theme and area

- **Used by** [Theme::get_properties\(\)](#)
- **Uses** [Theme::get_properties\(\)](#)

string function Theme::get_quickbottom([\$m = "]) [*line 534*]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct a list of quicklinks for top of page (if any)

(see also [get_quicktop\(\)](#)).

- Uses [Theme::get_quicklinks\(\)](#)

string function Theme::get_quicklinks(\$m, \$quick_section_id) [line 558]

Function Parameters:

- *string* **\$m** left margin for increased readability
- *string* **\$quick_section_id** the name of the property that holds the section containing these quicklinks

workhorse for constructing list of quicklinks

This creates HTML-code for links that can be displayed at the top of the page. These links are the pages (but not subsections) defined in the quicktop_section_id in \$this->config.

Note that this array may or may not exist and also that the section may or may not exist and that the section may or may not contain any visible pages. Mmm, that's a lot of may/maynot's...

Also note that these links are always displayed as text, even if a graphics image is defined in the corresponding node. The contents of the section can be found in \$this->tree.

- **TODO** should we take Apache's PATH_INFO feature into account to create friendly links?
- **Usedby** [Theme::get_quickbottom\(\)](#)
- **Usedby** [Theme::get_quicktop\(\)](#)

string function Theme::get_quicktop([\$m = ""]) [*line 521*]

Function Parameters:

- *string* **\$m** left margin for increased readability

construct a list of quicklinks for top of page (if any)

(see also [get_quickbottom\(\)](#)).

- Uses [Theme::get_quicklinks\(\)](#)

string function Theme::node2anchor(\$node_record, [\$attributes = NULL], [\$textonly = FALSE]) [*line 1087*]

Function Parameters:

- *array* **\$node_record** the node record to convert
- *array* **\$attributes** optional attributes to add to the HTML A-tag
- *bool* **\$textonly** if TRUE, no clickable images will be returned

construct an anchor from a node record

This constructs an array with key-value-pairs that can be used to construct an HTML anchor tag. At least the following keys are created in the resulting array: 'href', 'title' and 'anchor'. The latter is either the text or a referent to an image that is supposed to go between the opening tag and closing tag. Furthermore an optional key is created: target. The contents of the input array \$attributes is merged into the result.

If the parameter \$textonly is TRUE the key 'anchor' is always text. If \$textonly is NOT TRUE, the 'anchor' may refer to an image.

Note that the link text is always non-empty. If the node record has an empty link_text, the word 'node' followed by the node_id is returned. (Otherwise it will be hard to make an actual clickable link).

void function Theme::send_headers() [*line 157*]

send collected HTTP-headers to user's browser

This sends the headers that still need to be sent. These are collected in the array \$this-

>http_headers. If headers are already sent, this fact is logged (and the collected headers are not sent).

void function Theme::send_output() [line 183]

send collected output to user's browser

This first sends any pending HTTP-headers and subsequently outputs the page that is constructed by \$this->get_html()

void function Theme::set_preview_mode(\$is_preview_mode) [line 905]

Function Parameters:

- **\$is_preview_mode**

void function Theme::show_tree_walk([\$m = ""], \$subtree_id) [line 642]

Function Parameters:

- **\$m**
- **\$subtree_id**

Class TranslateTool

[line 56]

Methods to access properties of a language and modify translations

This class is used to manage languages and translations. The following functions are supplied

- add a new language
- edit the properties of an existing language (including active flag)
- add/edit translations of texts

The default action is to show a list of existing languages. From there the user can navigate to adding/editing language properties or manipulating translations in a particular language.

- **Package** wascore

TranslateTool::\$domains

array = array() [line 67]

- **Var** list of all language domains grouped by program, modules, themes and install

TranslateTool::\$languages

array = array() [line 64]

- **Var** list of all language records (including inactive ones), keyed with language_key

TranslateTool::\$output

object|null = NULL [line 58]

- **Var** collects the html output

TranslateTool::\$show_parent_menu

bool = FALSE [line 61]

- **Var** if TRUE the calling routing is allowed to use the menu area (e.g. show config mgr menu)

Constructor *void* function TranslateTool::TranslateTool(&\$output) *[line 77]*

Function Parameters:

- *object* **&\$output** collects the html output

construct a TranslateTool object

This initialises the TranslateTool and also dispatches the chore to do.

- **Uses** \$LANGUAGE
- **Uses** \$CFG

array function TranslateTool::a_param(\$chore, [\$language_key = NULL], [\$domain = NULL]) [*line 967*]

Function Parameters:

- *string* **\$chore** the next chore that could be done
- *string|null* **\$language_key** the language of interest or NULL if none
- *string|null* **\$domain** the full domain of interest or NULL if none

shorthand for the anchor parameters that lead to the translate tool

string function TranslateTool::code_highlight(&\$source, [\$highlight_on = ''], [\$highlight_off = '']) [*line 1103*]

Function Parameters:

- *string* **&\$source** the string that needs code highlighting
- *string* **\$highlight_on** is inserted before the code element that is highlighted
- *string* **\$highlight_off** is inserted after the code element that is highlighted

highlight code constructs in texts that are to be translated

this routine highlights the following code constructs:

- HTML-tages such as '' and ''
- Variables such as '{USERNAME}' and '{FILE}'
- Tildes in hotkeys such as '~Yes' and '~No'

All of these code elements are sandwiched between \$highlight_on and \$highlight_off.

The HTML-tags are escaped using `htmlspecialchars` making it possible to actually display them as text (otherwise they might be rendered as actual code in the browser). The HTML-codes '`<p>`' and '`
`' receive special treatment: they are rendered as visible text and also as a newline.

Note: This assumes that all '{' are eventually followed by a '}'. As long as this is true, we can easily use a `str_replace()` to sandwich {VARIABLE} between highlights. If there is only a single '{' or '}' the highlights won't match. It could be a problem and if it is, the relevant code should iterate / chomp chomp through the string with something like `ereg('([a-zA-Z0-9_]+)'), $string, $regs)`

As an added bonus, sequences of two consecutive spaces are replaced with non-breakable spaces. This is handy for phrases that use spaces to indent text, e.g. in simple text-only email messages.

By using a reference we prevent the endless coping of (long) strings to the stack; this should save time & space.

- **TODO** should we turn to `ereg()` instead of a simple `str_replace()` for {VARIABLE} highlighting?

bool function TranslateTool::diff_to_text(\$language_key, \$full_domain, &\$diff, &\$text) [line 1325]

Function Parameters:

- *string* **\$language_key** identifies the language
- *string* **\$full_domain** indicates the language domain
- *array* **&\$diff** contains all key-value-pairs for the modified translation
- *string* **&\$text** receives the complete sourcefile created from \$diff

convert an array with key-value-pairs to a php source file that can be included as a user translation

All key-value-pairs are converted to something like this:

```
...
$string['key'] = 'value';
...
```

We specifically use single quotes in order to prevent any variable expansion within the strings. We do escape embedded single quotes, naturally. Furthermore, some metadata is added to the top of the resulting file, including information about the creation time, the program version that was used, the version of the (English) source file on

which this translation is based and finally information about the file version of the system strings which was used to diff against.

Note: If the file with these system strings do not exist (because the language is all new, indicated by a version 'v0', `_all_` strings are stored in the diff and thus in the user file. That file could be used as a new starting point for the new language in a next version of the program.

Note: we try very hard to defeat tricks with the contents of the metadata (i.e. we don't trust `_full_name` and `_email` to not contain tricks like `'*' followed by '/'` (which would prematurely end the comment in the header) etc.

array function TranslateTool::get_dialogdef_language([\$language_key = ""]) [line 704]

Function Parameters:

- *string* **\$language_key** identifies the language to edit (empty string for add new language)

construct the language dialog (used for both add and edit)

this constructs a language dialog definition, maybe filled with data The main difference between dialogs for add and edit is that an existing language code (`$language_key`) cannot be changed; the corresponding field is shown in 'viewonly' mode. Another small difference is that an existing language cannot have itself as a parent language.

Note that we populate the 'edit' dialog with existing data from `$this->languages`.

array function TranslateTool::get_dialogdef_language_domain([\$language_key = "], [\$full_domain = "]) [line 783]

Function Parameters:

- *string* **\$language_key** identifies the language to edit
- *string* **\$full_domain** identifies the language domain

construct the translation dialog for selected language and domain

this constructs a translation dialog definition, filled with translations for language `$language_key` and domain `$full_domain`. The labels for the fields are derived from the English texts in `$full_domain`, in the order specified by the English file. If the English file contains comments, these are added to the item too (to be displayed as additional information for the translator). The current translation of the `$full_domain` is retrieved the usual way, via function `t()` (shorthand for `$LANGUAGE->get_phrase()`) but without translating any variables (e.g. `{VALUE}`). Note that if a translation of a phrase does not exist in the target language, the `get_phrase()` routine will eventually yield the English translation (after trying the language parents first). Also note that the translated phrases could be retrieved from a user file (ie. a file from `$CFG->datadir/languages/$language_key/$full_domain`).

- **TODO** try to figure this out: when the delimiter in \$name was a dot '.' \$_POST contained a '_' instead. WTF? (it seems that a colon works... for now)
- **Uses** \$USER
- **Uses** \$CFG

array function TranslateTool::get_domains() [line 995]

return an ordered list of translation domains

this constructs a list of language domains, grouped by 'program','modules','themes' or 'install'. This array is the basis for validating full domains (in \$_POST'ed data) and also to construct a menu.

Note that we use the translations from the files themselves in the current language to construct this list. Every translatefile should have at least the string 'translatetool_title' and 'translatetool_description'. Currently the sort order is based on the (internal) name of the modules. This should do the trick for translators: the order of files to translate in the menu does not depend on the translation of the module- or theme-title. (In the page manager and elsewhere it may be different).

string function TranslateTool::get_icon_edit(\$language_key) [line 940]

Function Parameters:

- *string* **\$language_key**

construct a clickable icon to edit the properties of this language

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

array function TranslateTool::get_options_languages([\$skip_language_key = ""]) [line 878]

Function Parameters:

- *string* **\$skip_language_key** suppress this language in list (language cannot be its own parent)

fetch a list of languages available as parent language

this constructs a list of languages that can be used as a list of parent language options in a listbox or radiobuttons.

void function TranslateTool::get_strings_system(\$language_key, \$full_domain, &\$string, &\$comment, \$languagekey, \$full_domein) [*line 1151*]

Function Parameters:

- *string* **\$languagekey** the two or three letter ISO 639 language code
- *string* **\$full_domein** the language domain of interest
- *array* **&\$string** receives the translations (this parameter must be called 'string')
- *array* **&\$comment** receives the comments (this parameter must be called 'comment')
- **\$language_key**
- **\$full_domain**

retrieve strings (translations) and comments from an official (system) translation file

This routine reads the system translations for \$language_key and \$full_domain from a file. For the translations of the main program we look for a single file in \$CFG->progdire/lanquages/\$language_key/. For modules, themes and addons we try two different locations: one within the moduel/theme/addon directory tree and subsequently in the generic directory \$CFG->progdire/lanquages/\$language_key/.

The names of modules/themes/addons are derived by stripping the 2-character prefix (m_, t_ or a_) from the full domain.

Translations for the installer are searched for in the /program/install/lanquages tree.

int function TranslateTool::guess_row_count(&\$text, [\$maximum = 15]) [*line 1056*]

Function Parameters:

- *string* **&\$text** the string to analyse
- *int* **\$maximum** the maximum value this routine returns

try to calculate a reasonable number of textarea rows based on the contents of \$text

By using a reference we prevent the endless coping of (long) strings to the stack; this should save time & space.

void function TranslateTool::languages_overview() [line 175]

display list of languages with edit icons and an option to add a language

this constructs the languages overview: a link to add a language, followed by a list of languages based on the languages in the database. Every language has an icon through which the properties of the language can be modified, including setting/resetting the active flag. (Only active languages can be used on the website and in the CMS). Note that we use `_all_` languages here, including inactive ones.

Note that the calling routine (the tools manager) is allowed to display a menu because we set the parameter `show_parent_menu` to `TRUE` here.

The constructed list looks something like this:

```
    Add a language
[E] Deutsch (de) (inactive)
[E] English (en)
[E] Nederlands (nl)
...
```

The clickable icons [E] lead to the Edit language properties. The clickable titles lead to the actual translations The clickable link 'Add an area' leads to the add new language dialog.

- **TODO** should we add a paging function to the (perhaps looooong) list of languages?
- **Uses** `$WAS_SCRIPT_NAME`
- **Uses** `$CFG`
- **Uses** `$USER`

void function TranslateTool::language_add() [line 232]

present the language dialog where the user can enter properties for a new language

this displays a dialog where the user can enter the properties of a new language. These properties are:

- name (expressed in the language itself)
- key (2- or 3-letter code, presumably based on ISO 639-1 or ISO 639-2)
- parent_key (this language is based on which existing language)

- active flag

The new language is saved via performing the 'chore' TRANSLATETOOL_CHORE_LANGUAGE_SAVE_NEW.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER

void function TranslateTool::language_edit() [line 253]

show the language edit dialog

display a dialog where the user can modify language properties. we re-use the routine that created the add language dialog.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER

void function TranslateTool::language_save() [line 419]

validate and save modified data to database

this saves data from the edit language dialog if data validates. If the data does NOT validate, the edit screen is displayed again otherwise the languages overview is displayed again.

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$LANGUAGE
- **Uses** \$CFG

void function TranslateTool::language_savenew() [*line 301*]

save the newly added language to the database

This saves the essential information of a new language to the database, using sensible defaults for the other fields. Also, a data directory is created in \$CFG->datadir

If something goes wrong, the user can redo the dialog, otherwise we return to the languages overview, with the newly added language in the list, too.

Apart from the standard checks the following checks are done:

- the language key should be an acceptable directory name
- the language key should be lowercase
- the language key should not exist already (in \$this->languages)
- the directory should not yet exist
- the directory must be created here and now

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$LANGUAGE
- **Uses** \$CFG

bool function TranslateTool::put_strings_userfile(\$language_key, \$full_domain, &\$diff) [*line 1259*]

Function Parameters:

- *string* **\$language_key** identifies the language to save
- *string* **\$full_domain** indicates which language domain needs to be saved
- *array* **&\$diff** contains all key-value-pairs for the modified translation

save new or changed translations to a file under CFG->datadir/languages

array function TranslateTool::render_translation_dialog(\$href, &\$dialogdef, [\$method = 'post'], [\$attributes = '']) [*line 1204*]

Function Parameters:

- *string* **\$href** the target of the HTML form

- *array* **&\$dialogdef** the array which describes the complete dialog
- *string* **\$method** method to submit data to the server, either 'post' or 'get'
- *string|array* **\$attributes** holds the attributes to add to the form tag

render a translation dialog based on a dialog definition

This routine looks a bit like the generic [dialog_quickform\(\)](#). The differences are:

- we show a comment (if any) in a box before label and input
- the labels don't have hotkeys based on tildes at all (except the submit buttons)
- comments and labels are wrapped in separate div's especially for the occasion

We do take any errors into account: fields with errors are displayed using the additional error class (which shows a label completely in red to indicate the error).

- Uses [html_form\(\)](#)

void function TranslateTool::show_domain_menu(\$language_key, [\$current_domain = '']) [*line 905*]

Function Parameters:

- *string* **\$language_key** the language currently being edited
- *string* **\$current_domain** the currently selected language domain (used to emphasize the option in the menu)

display the domain menu via \$this->output

This displays a clickable menu on in the menu area on the left of the screen.

bool function TranslateTool::show_parent_menu() [*line 132*]

allow the caller to use the menu area (or not)

this routine tells the caller if it is OK to use the menu area (TRUE returned) or not (FALSE returned).

bool function TranslateTool::submit_diff_to_project(\$language_key, \$full_domain, &\$diff) [*line 1384*]

Function Parameters:

- *string* **\$language_key** identifies the language to submit
- *string* **\$full_domain** indicates which language domain needs to be submitted
- *array* **&\$diff** contains all key-value-pairs for the modified translation

send new or changed translations back to the project

This sends an e-mail back to the project with the translation. We do so in the form of an attachment, but with a 'safe' extension (.bin rather than .php). This means that we will be able to traverse any firewalls and spamfilters and malware detectors.

The `_notes` are used as the body of the message, the file is attached.

Note that we send a copy of the message to the site itself (either the from-address or the reply-to-address).

void function TranslateTool::translation_edit() [line 490]

show an edit dialog with phrases from \$full_domain in \$language_key

After some sanity checking this routine shows a dialog where the user can edit translations for the selected language and domain. Note that this could be a huge dialog, depending on the size of the language domain ('admin' is notoriously large). Sending this routine to the browser can take some time.

void function TranslateTool::translation_save() [line 551]

save the modified translations in a file in the tree CFG->datadir/languages/

this routine validates the dialog data and attempts to save the changes in the file `$full_domain` in the directory `CFG->datadir/languages/$language_key/`. Also, we may need to send a message to the Website@School project with our changes (depending on the flag `_submit`).

Class Useraccount

[line 251]

Methods to access properties of the account of the logged in user

This deals mainly with retrieving information about the user that is currently logged in. There is one exception: a user that is NOT logged in can still have a `$USER` object, but there are no privileges in that case. The special `user_id` in that case is 0.

The constructor reads the important data from the database. This includes things like the full name of the user and the email address. This information is stored in the object and can be used, e.g. `$USER->email`. This information is basically copied from the table 'users'.

Furthermore, any properties for this user are retrieved from the table 'users_properties'. All properties are stored in an array. These can be used directly via `$USER->properties['foobar']`.

Access Control

Finally we deal with access control. This has become quite complex but still manageable (I hope). There are six tables dealing with acl's:

- `acls`: site-wide permissions for jobs, intranet, modules and nodes
- `acls_areas`: permissions for intranet, modules and nodes at the area level
- `acls_nodes`: permissions for modules and nodes at the node level
- `acls_modules`: permissions for modules at the site level
- `acls_modules_areas`: permissions for modules at the area level
- `acls_modules_nodes`: permissions for modules at the node level

The user has at least one associated ACL: the `acl_id` field in the user record. Additional ACLs are associated with the user via group memberships. All ACLs are integer bitmasks where a '1' grants a permission for something and a '0' denies permission.

All bits '0' is a special case: this is the default (nothing allowed) and hence does not have to be stored: the mere non-existence of permissions implies no permissions.

All bits '1' is also a special case, dubbed 'ROLE_GURU'. If an ACL has this value, it means that all current (and future) permissions are granted. A user with `ROLE_GURU` can do anything.

Of the six tables, only the first one (`acls`) is read immediately in the constructor. The others are read on demand. This is done by initially setting the corresponding cache variable to `NULL`. If the table has been read, the variable will always be of type 'array', even though that array may be empty (indicating no permissions).

The permissions from the ACLs are combined between the user's `acl` and the optional group-`acls`. Only the combination of user and group permissions is cached in order to save space. This is done by OR'ing the permission bits. Note that the condition all bits '0' is not stored, also to save space.

There are some functions to test for individual permissions:

- `has_site_permissions()`
- `has_area_permissions()`
- `has_node_permissions()`
- `has_job_permissions()`
- `has_intranet_permissions()`
- `has_module_site_permissions()`

- `has_module_area_permissions()`
- `has_module_node_permissions()`

Example: in order to determine whether a user has access to the intranet in area #2, the following could be used:

```
$area_id = 2;
if ($USER->has_intranet_permissions(ACL_ROLE_INTRANET_ACCESS,$area_id)) {
    ....
}
```

The effect of this call is as follows. First the routine checks the (already cached) intranet-permissions at the site level. If access is granted at the site level, there is no need to look any further because obviously this user has access to this intranet (private area) and all other current and future intranets. If not, the routine looks at intranet permissions at the area level. The first time this will trigger reading and caching the table for area-level permissions. In this case (intranet-access), the area-level permissions provide the definitive go/nogo for this user (there is no point in having intranet-access-permissions at the node level).

Note that the 'lower' ACL is only checked if the 'higher' does not provide answers. This saves unnecessary trips to the database.

Note that this works much the same for the other `has_XXX_permissions()`: first the site-level is tried, then the area-level and finally the node-level (when applicable).

ACLs for modules

Access to the CMS itself is fairly fine-grained. The permissions are stored in the fields 'permissions_nodes' in the tables 'acls' (site-level), 'acls_areas' (area-level) and 'acls_nodes' (node-level). These permissions basically deal with the page manager (the piece de resistance of the whole system).

However, there are modules that can be linked to nodes, e.g. a chat or a forum or an agenda which also require authorised users and permissions. These permissions are stored in three tables: 'acls_modules' (site-level), 'acls_modules_areas' (area-level) and 'acls_modules_nodes' (node-level). This works pretty much the same as the permissions for the CMS itself, but it that there is an extra parameter, namely the `module_id`.

Once again, the permissions are only read when necessary. I.e., if the site-level already grants a permission, the area and node level are not read from the database. This saves time and space.

Roles and permissions

Permissions are individual flags that allow or disallow a certain feature, e.g. 'adding a page to a section'. In order to keep these permissions manageable groups of permissions are combined yielding a limited number of 'roles'. A 'role' is a combination of 1 or more permission bits. Assigning permissions (in the user account manager) is done by assigning these 'roles' to a user, either sitewide, areawide or per node. These roles are dubbed sitemaster, areamaster, sectionmaster, pagemaster and contentmaster. The 'higher' roles incorporate the 'lower' roles: permissions of a sectionmaster include those of a pagemaster and a contentmaster.

- **Package** wascore

Useraccount::\$acls

```
array = array('permissions_jobs' => ACL_ROLE_NONE,  
             'permissions_intranet' => ACL_ROLE_NONE,  
             'permissions_modules' => ACL_ROLE_NONE,  
             'permissions_nodes' => ACL_ROLE_NONE) [line 277]
```

- **Var** \$acIs contains the highest-level (site level) permissions, cached from table acIs

Useraccount::\$acIs_areas

```
null|array = NULL [line 283]
```

- **Var** \$acIs_areas holds area-level permissions, cached from acIs_areas

Useraccount::\$acIs_modules

```
null|array = NULL [line 289]
```

- **Var** \$acIs_modules holds site-level permissions for modules, cached from acIs_modules

Useraccount::\$acIs_modules_areas

```
null|array = NULL [line 292]
```

- **Var** \$acIs_modules_areas holds area-level permissions for modules, cached from acIs_modules_areas

Useraccount::\$acIs_modules_nodes

```
null|array = NULL [line 295]
```

- **Var** \$acIs_modules_nodes holds node-level permissions for modules, cached from

acIs_modules_nodes

Useraccount::\$acIs_nodes

null|array = NULL [line 286]

- **Var** \$acIs_nodes holds node-level permissions, cached from acIs_nodes

Useraccount::\$acI_id

int = 0 [line 271]

- **Var** \$acI_id identifies the main acI for this user

Useraccount::\$area_permissions_from_nodes

array|null = NULL [line 310]

- **Var** cache for admin permissions based on node permissions

Useraccount::\$editor

string = [line 304]

- **Var** \$editor the preferred editor for this user (empty implies system default from \$CFG->editor)

Useraccount::\$email

string = [line 262]

- **Var** \$email

Useraccount::\$full_name

string = [line 259]

- **Var** \$full_name

Useraccount::\$high_visibility

bool = FALSE [line 301]

- **Var** \$high_visibility

Useraccount::\$is_logged_in

bool = FALSE [line 307]

- **Var** \$is_logged_in TRUE if user is logged in, FALSE otherwise

Useraccount::\$language_key

string = [line 265]

- **Var** \$language_key

Useraccount::\$path

string = [line 268]

- **Var** \$path directory that holds the personal data files relative to "{\$CFG->datadir}/users/"

Useraccount::\$properties

array = array() [line 298]

- **Var** \$properties

Useraccount::\$related_acls

array = array() [line 274]

- **Var** \$related_acls holds acl_id -> groupname/capacity pairs related to this user

Useraccount::\$username

string = [line 256]

- **Var** \$username

Useraccount::\$user_id

int = 0 [line 253]

- **Var** \$user_id

Constructor *void* function Useraccount::Useraccount([\$user_id = 0]) [line 318]

Function Parameters:

- *int* **\$user_id** identifies data from which user to load, 0 means no user/a passerby

get pertinent user information in core

array function Useraccount::fetch_acls_from_table(\$table, \$where) [*line 567*]

Function Parameters:

- *string* **\$table** name of the table which holds the acls
- *string* **\$where** a ready-to-use whereclause that selects the relevant acls based on acl_id

retrieve acl-data from table into a sparse array

bool function Useraccount::has_area_permissions(\$mask, \$area_id, [\$field = 'permissions_nodes']) [*line 407*]

Function Parameters:

- *int* **\$mask** bitmap of OR'ed permissions to test for
- *int* **\$area_id** which area to test
- *string* **\$field** name of permissions to check (default 'permissions_nodes')

determine user's permissions for an area

this looks at the area-level permissions for manipulating nodes and areas. However, we first look at the site-level permissions. If those already satisfy the request, we return immediately. If not, the permissions are fetched from the table acls_areas or from the cached data. We only fetch the data if it is really necessary.

bool function Useraccount::has_intranet_permissions(\$mask, \$area_id) [*line 475*]

Function Parameters:

- *int* **\$mask** bitmap of OR'ed permissions to test for
- *int* **\$area_id** which area to test

determine user's permissions for an intranet area

this looks at the area-level permissions for intranet areas.

bool function Useraccount::has_job_permissions(\$mask) [*line 459*]

Function Parameters:

- *int* **\$mask** bitmap of OR'ed permissions to test for

determine user's permissions for a job

bool function Useraccount::has_node_permissions(\$mask, \$area_id, \$node_id, [\$field = 'permissions_nodes']) [*line 435*]

Function Parameters:

- *int* **\$mask** bitmap of OR'ed permissions to test for
- *int* **\$area_id** which area to test
- *int* **\$node_id** which node to test
- *string* **\$field** name of permissions to check (default 'permissions_nodes')

determine user's permissions for a node within an area

bool function Useraccount::has_site_permissions(\$mask, [\$field = 'permissions_nodes']) [*line 382*]

Function Parameters:

- *int* **\$mask** bitmap of OR'ed permissions to test for
- *string* **\$field** name of permissions to check (default 'permissions_nodes')

determine user's permissions for the site-level

this looks at the site-level permissions for manipulating nodes and areas etc. The permissions are cached from the table acls.

bool function Useraccount::is_admin() [*line 492*]

determine whether the user has administrator privilege

If this user has access to the admin startcenter, she is considered an administrator. Further access depends on the other bits in the job permissions, but at least she is allowed to enter the system via admin.php.

bool function Useraccount::is_admin_pagemanager(\$area_id) [*line 516*]

Function Parameters:

- *int* **\$area_id** the area to examine

determine whether the user has administrator privilege for pagemanager

This routine determines whether a user has any privileges at all for the page manager. This is true in the following cases:

- the user has sitewide permissions that belong to one of the roles contentmaster, pagemaster, sectionmaster or areamaster, OR
- the user has areawide permissions for one of those roles, OR
- the user has permissions for one of those roles in at least one node in the requested area.

The calculations in the third case are cached for all areas.

string function Useraccount::where_acl_id([\$field = 'acl_id']) [*line 615*]

Function Parameters:

- *string* **\$field** identifies the fieldname to check

a convenient routine to construct a selection of acls

this constructs a where clause of the form '(acl_id = 1) OR (acl_id = 2) OR (acl_id = 3)'

Class UserManager

[*line 40*]

User management

- **Package** wascore

- **TODO** Perhaps this class should be merged with the GroupManager class because there is a lot of overlap. Mmmmm.... maybe in a future refactoring operation.

UserManager::\$output

object|null = NULL [line 42]

- **Var** collects the html output

UserManager::\$show_parent_menu

bool = FALSE [line 45]

- **Var** if TRUE the calling routing is allowed to use the menu area (e.g. show account mgr menu)

Constructor *void* function UserManager::UserManager(&\$output) [line 54]

Function Parameters:

- *object* **&\$output** collects the html output

construct a UserManager object

This initialises the UserManager and also dispatches the task to do. This also loads the loginlib: we need that in order to manipulate the user password.

array|bool function UserManager::areas_expand_collapse(\$areas_open, \$area_id) [line 2173]

Function Parameters:

- *array|bool* **\$areas_open** current state of indicator(s) for 'open' and 'closed' areas
- *int|null* **\$area_id** the area to expand/collapse or NULL if nothing needs to be done

manipulate the current state if indicator(s) for 'open' and 'closed' areas

this manipulates the current state of 'open' and 'closed' areas in \$areas_open. If \$area_id is NULL, we don't have to do anything but simply return the current state. If \$area_id is 0 (zero), we need to toggle all areas at once (area_id = 0 implies the site level toggle) If \$area_id is an integer, it is assumed to be a valid area_id and that area should be toggled.

array function UserManager::a_params([\$task = NULL], [\$user_id = NULL], [\$module_id = NULL]) [line 1665]

Function Parameters:

- *string|null* **\$task** the next task to do or NULL if none
- *int|null* **\$user_id** the user of interest or NULL if none
- *int|null* **\$module_id** the module of interest or NULL if none

shorthand for the anchor parameters that lead to the user manager

void function UserManager::calc_acl_id(\$user_id) [line 2138]

Function Parameters:

- **\$user_id**

bool function UserManager::delete_user_acl(\$user_id) [line 1361]

Function Parameters:

- *int* **\$user_id** the key to the user account to delete

remove all records relating to a single acl_id from various acl-tables

this bluntly removes all records from the various acls* tables for the acl_id of this user and subsequently other records associated with this user. Whenever there's an error deleting records, the routine bails out immediately and returns FALSE. If all goes well, TRUE is returned. Any errors are logged, success is logged to DEBUG-log.

array function UserManager::get_dialogdef_add_user() [line 1859]

construct the add userdialog

array function UserManager::get_dialogdef_add_usergroup(\$user_id) [line 1273]

Function Parameters:

- *int* **\$user_id** limit the options to groups this user is NOT already a member of

construct a dialogdef for selecting a group/capacity

bool|array function UserManager::get_dialogdef_edit_user(\$user_id) [*line 1944*]

Function Parameters:

- *int* **\$user_id** indicates which user to edit

construct the edit user dialog

- Uses \$LANGUAGE

array function UserManager::get_editor_names() [*line 2103*]

prepare a list of available editors

- **TODO** this list here is hardcoded: we do not expect to be adding or removing editors to/from the CMS soon. However, it might be cleaner to do this elsewhere.

string function UserManager::get_fname(\$item) [*line 2156*]

Function Parameters:

- *array* **\$item** contains definition of a single field in a dialog

shorthand for the first readable name in a dialogdef item

string function UserManager::get_icon_delete(\$user_id) [*line 1779*]

Function Parameters:

- *int* **\$user_id** the user to delete

construct a clickable icon to delete this user

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

string function UserManager::get_icon_edit(\$user_id) [*line 1806*]

Function Parameters:

- *int* **\$user_id** the user to edit

construct a clickable icon to edit the properties of this user

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

string function UserManager::get_icon_groupdelete(\$user_id, \$group_id) [*line 1835*]

Function Parameters:

- *int* **\$user_id** the group to delete
- **\$group_id**

construct a clickable icon to delete a membership from this user

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$USER
- **Uses** \$CFG

int function UserManager::get_num_user_records(\$group_id) [*line 1742*]

Function Parameters:

- *int* **\$group_id** which group needs to be counted

calculate the total number of users in a specific group

this calculates the total number of users in group \$group_id. If \$group_id equates to GROUP_SELECT_ALL_USERS, the grand total is returned, if it equates to GROUP_SELECT_NO_GROUP the number of users without a group is calculated.

- **Uses** \$DB

array function UserManager::get_options_available_groups_capacities(\$user_id) [*line 1318*]

Function Parameters:

- *int* **\$user_id** the user to which this list of available groups applies

construct a list of groups still available for this user

this constructs an array with available groups/capacities for the user \$user_id. If the user is already a member of all available groups or there are no groups at all, the list consists of a single option 'No groups available'.

The values in this list are constructed from the primary key values of the underlying groups_capacities table. These two numbers (group_id and capacity_code) are separated with a colon ':' to make it easier to parse once we are to save the values (in the table users_groups_capacities).

The SQL-statement looks quite complex. What it does is using the table groups_capacities as

a starting point for `_all_ valid` (ie `capacity_id != CAPACITY_NONE`) combinations of group and capacity. By left-joining the table `users_groups_capacities` with a very specific `ON`-clause, and leaving out the column `capacity_code`, the resulting list consists of all combinations of group and capacity but without any entries that have a group of which the user is already a member, no matter what capacity. In other words: if a user is already a member of a group with capacity A, this user cannot be member of the same group with capacity B. Finally, the table `groups` is used to retrieve the group information such as the `groupname` and the `active-flag`.

The resulting list is ordered by `groupname` and subsequently by the `sort_order` of the `capacity_code`. However, inactive groups are sorted after the active groups so they appear near the bottom of the list.

`array` function `UserManager::get_user_names($user_id)` [line 2090]

Function Parameters:

- `int $user_id` identifies the user of interest

shortcut to retrieve the username and full name of the selected user

`array|bool` function `UserManager::get_user_records($group_id, $limit, $offset)` [line 1697]

Function Parameters:

- `int $group_id` selection for users
- `int $limit` maximum number of records to retrieve
- `int $offset` number of records to skip in result set

retrieve (a selection of) all user records from the database

this retrieves a subset of all existing user accounts from the database. The selection depends on the value of `$group_id`:

- `$group_id == GROUP_SELECT_ALL_USERS (-1)`: all users ordered by active, username
- `$group_id == GROUP_SELECT_NO_GROUP (0)`: all users without a group, ordered by active, username
- otherwise: users in group `$group_id`, ordered by active, username

Note that in the first two cases there is no capacity, in the third case every user has capacity relating to the specified group.

bool function UserManager::has_job_permission(\$user_id, \$job) [*line 2127*]

Function Parameters:

- *int* **\$user_id** group to check
- *int* **\$job** job a bitmask indicating a particular job

determine whether a user has permissions for a particular job

this determines whether this user has permissions to access the specified job, e.g. do they have access to the page manager. If so, we can display the menu option, otherwise we can suppress it and keep the menu clean(er).

void function UserManager::show_breadcrumbs_adduser() [*line 1648*]

display breadcrumb trail that leads to the add new user dialog

- **Uses** [UserManager::show_breadcrumbs_overview\(\)](#)
- **Uses** \$WAS_SCRIPT_NAME;

void function UserManager::show_breadcrumbs_overview() [*line 1604*]

display breadcrumb trail that leads to users overview screen

- **Usedby** [UserManager::show_breadcrumbs_adduser\(\)](#)
- **Usedby** [UserManager::show_breadcrumbs_user\(\)](#)
- **Uses** \$WAS_SCRIPT_NAME;

void function UserManager::show_breadcrumbs_user(\$user_id) [*line 1631*]

Function Parameters:

- *int* **\$user_id** the user of interest

display breadcrumb trail that leads to the edit user dialog

- Uses [UserManager::show_breadcrumbs_overview\(\)](#)
- Uses \$WAS_SCRIPT_NAME;

void function UserManager::show_menu_overview(\$group_id) [line 1524]

Function Parameters:

- *int|null* **\$group_id** identifies the current selection

display a menu showing groups of users (if any) + corresponding breadcrumb trail

this constructs a list of links allowing for a quick selection of a subset of users This looks a little like this:

All users (66)
No group (5)
faculty (14)
grade12 (7)
...
webmasters (2)

The indication of the current selection in the menu is based on \$group_id. Most of the time this is a genuine group_id. However, 'All users' and 'No group' are special cases:

- The value GROUP_SELECT_ALL_USERS (-1) cannot be a genuine group_id because these are always > 0.
- The value GROUP_SELECT_NO_GROUP (0) cannot be a genuine group_id because these are always > 0.

- Uses \$DB;
- Uses #AS_SCRIPT_NAME

void function UserManager::show_menu_user(\$user_id, [\$current_task = NULL], [\$current_module_id = NULL])
[line 1419]

Function Parameters:

- *int* **\$user_id** identifies the user
- *string* **\$current_task** the task to show highlighted
- *int* **\$current_module_id** the current module to show highlighted

show the user menu with current option highlighted

this constructs the user menu. Only the relevant options are displayed (eg. if the user is not an admin, no pagemanager option is displayed).

void function UserManager::show_parent_menu() [line 134]

void function UserManager::users_overview() [line 184]

display a list of existing users and an option to add a user

This constructs the heart of the user manager: a link to add a user, followed by a list of links for deleting an modifying selected (see below) users. The list of users is ordered as follows. First the active users are displayed, an after that the inactive users are displayed. The sort order is based on the short name of the user.

Note that a selection is made of all user accounts, based on a choice the user makes from the menu (see [show_menu_overview\(\)](#)). This list to show is selected as follows:

- if the parameter 'group' is NOT set in \$_GET[] and this is the 1st time, all users are listed (equivalent with GROUP_SELECT_ALL_USERS). If we are returning, the \$_SESSION may contain another default group selectiond
- if the parameter 'group' is set to GROUP_SELECT_ALL_USERS (-1), all users are listed
- if the parameter 'group' is set to GROUP_SELECT_NO_GROUP (zero), all users without a group are listed
- if the parameter 'group' has another value, the users of that group are listed

The list of existing users is paginated, ie. if there are more than a screenfull, an additional paginator is displayed at the end of the list. The screen always starts with an add a user link though.

Note that the list of existing users shows the full name and the username in parenthese. If a 'real' group is selected (ie. not the collection of users without a group or all users), the capacity of that user in that group is also displayed.

Example: Amelia Cackle, a 'Principal' in the 'faculty' group, is displayed like this in the faculty group: Amelia Cackle (acackl) (Principal)

void function UserManager::user_add() [line 313]

present 'add user' dialog where the user can enter minimal properties for a new user

this displays a dialog where the user can enter the minimal necessary properties of a new user. These properties are:

- name (e.g. 'hparkh')
- full name (e.g. 'Helen Parkhurst')
- a password
- an e-mail address
- the active flag

Other properties will be set to default values and can be edited later on by editing the user account.

The new user is saved via performing the task TASK_USER_SAVE_NEW

- **Uses** \$WAS_SCRIPT_NAME

void function UserManager::user_admin() [line 1156]

show a dialog for modifying admin permissions for a user

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG

void function UserManager::user_delete() [line 770]

delete a user after confirmation

this either presents a confirmation dialog to the user OR deletes a user with associated acs.

Note that this routine could have been split into two routines, with the first one displaying the confirmation dialog and the second one 'saving the changes'. However, I think it is counter-intuitive to perform a deletion of data under the name of 'saving'. So, I decided to use the same routine for both displaying the dialog and acting on the dialog.

- **TODO** should we also require the user to delete any files associated with the user before we even consider deleting it? Or is it OK to leave the files and still delete the user. Food for thought.
- **TODO** since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

void function UserManager::user_edit() [line 505]

present an 'edit user' dialog filled with existing data

- **TODO** maybe it is better to call this routine with \$user_id as a parameter? that allows for moving from adduser() -> saveuser() -> edituser(\$user_id). Mmmm, food for thought

void function UserManager::user_groupadd() [line 940]

present 'add membership' dialog

this displays a simple dialog where the user can add a membership to a user account, one at a time. Basically we show a picklist with all available group/capacity-combinations. Here "available" means:

- only groups of which the user is currently NOT a member
- only non-0 group/capacity-combinations that occur in the groups_capacities_table (capacity 0 implies: no capacity)

An additional feature is that the user can become member of inactive groups. However, these groups are sorted at the end of the picklist.

void function UserManager::user_groupdelete() [line 1068]

end the group membership for the selected user

- **Uses \$DB**

void function UserManager::user_groups() [line 865]

present an overview of group memberships for the specified user

this constructs a link to add a membership to the user account and a list of existing memberships, if any, including a delete button per membership.

The SQL-query retrieves the list of existing memberships from the database, ordered by the short groupname. The data is validated by joining to the table groups_capacities. If for some reason there exists an invalid combination of group_id and capacity_code in users_groups_capacities table, it will not show up in the list here.

Note that it is currently not possible to change a users' group membership, i.e. you cannot promote a user from 'Member' to 'Chair' for a group: you have to delete the group membership first, and subsequently add it again with the correct capacity.

void function UserManager::user_groupsave() [line 976]

save the new group/capacity for the selected user

this adds a record to the users_groups_capacities table, indicating the group membership and the corresponding capacity for the user.

- **Uses** \$WAS_SCRIPT_NAME

void function UserManager::user_intranet() [line 1114]

show a dialog for modifying intranet permissions for a user

- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG

void function UserManager::user_pagemanager() [line 1197]

show a dialog for modifying page manager permissions for a user

- **Uses** \$WAS_SCRIPT_NAME

- **Uses** \$CFG

void function UserManager::user_save() [*line 532*]
save edited user data to the database

data function UserManager::user_savenew() [*line 337*]
save a new user to the database

this saves a new user to the database. This involves at least two tables: a record in the users table with basic information and also a record with access control in the acls table.

- **TODO** shouldn't we end with the edit-user dialog rather than the users overview? that might make more sense...
- **TODO** maybe we should find a more elegant way to check a field for uniqueness
- **Uses** \$WAS_SCRIPT_NAME
- **Uses** \$CFG

void function UserManager::user_save_basic(\$user_id) [*line 648*]
Function Parameters:

- *int* **\$user_id** the account to save (pkey in users table)

save basic properties of user account

- **Uses** \$WAS_SCRIPT_NAME

Class Zip

[line 139]

Create simple and compatible ZIP-archives

With this class it is possible to create ZIP-archives that are compatible with the original PKZip 2.04g. This class does not provide a way to read ZIP-archives.

There are three possible options for the output:

- write the ZIP-archive directly to a file (OpenZipfile())
 - output ('stream') directly to the user's browser, including appropriate headers (OpenZipstream())
 - collect the output in a buffer in memory (OpenZipbuffer()).
There are two different ways to add to the ZIP-archive:
 - add a file from the filesystem (AddFile())
 - add data from memory as if it was a file (AddData())
- The ZIP-archive needs to be closed before it is useable (CloseZip()).

Special features:

- it is not necessary to manually add a directory to the ZIP-archive because all directories that lead to a file will be added automatically
 - both AddFile() and AddData() allow for on-the-fly (re)naming; i.e. the name of the file in the ZIP-archive can be different from the name of the file in the filesystem
 - it is possible to add a comment to an individual file
 - it is possible to add a comment to the ZIP-archive
- Limitations

This class might use a lot of memory when creating ZIP-archives, especially the ZIP_TYPE_BUFFER variant which eventually requires the size of the resulting ZIP-archive plus (worst case) the size of the largest file plus the size of the largest compressed file. This might be a problem with large files or many, many smaller files. A workaround could be to either stream the ZIP-archive directly (ZIP_TYPE_STREAM) or write to a file (ZIP_TYPE_FILE) because those variants only require the size of the largest file, the largest compressed file and the size of the central directory.

This class is not able to read ZIP-archives.

This class either stores a file as-is using the PKZIP 'Store' method or compresses the file using the 'Deflate' method. There is no support for other (more advanced) compression algorithms and no encryption is used.

References

I implemented this class using the following references.

[1] The ultimate definition of the ZIP-archive format as published by PKWare, Inc. See: <http://www.pkware.com/appnote.txt> or <http://www.pkware.com/support/zip-application-note>. I used version 6.3.2 which was published on 28 September 2007.

[2] RFC1950 - ZLIB Compressed Data Format Specification version 3.3, P. Deutsch, J-L. Gailly (May 1996), <http://www.faqs.org/rfcs/rfc1950>

[3] RFC1951 - DEFLATE Compressed Data Format Specification version 1.3, P. Deutsch (May 1996), <http://www.faqs.org/rfcs/rfc1951>

[4] Disk Operating System Technical Reference, IBM Corporation 1985, Chapter 5 (DOS Disk Directory).

[5] Official registration of the application/zip MIME-type:
<http://www.iana.org/assignments/media-types/application/zip>

Examples

Typical usage of this class is as follows.

```
Example 1 - store 3 existing files in a ZIP-archive $zip = new Zip;
$zip->OpenFile("/tmp/test.zip");
$zip->AddFile("/tmp/foo.txt");
$zip->AddFile("/tmp/bar.txt");
$zip->AddFile("/tmp/baz.txt");
$zip->CloseZip();
```

```
Example 2 - store a chunk of data in a ZIP-archive in memory $zip_archive = "";
$data = "This is example-data that ends up in file QUUX.TXT";
$zip = new Zip;
$zip->OpenBuffer($zip_archive);
$zip->AddData($data,'QUUX.TXT');
$zip->CloseZip();
```

```
Example 3 - directly stream a file in a ZIP-archive and rename on the fly $zip = new Zip;
$zip->OpenStream('htdocs.zip');
$zip->AddFile("/var/www/index.html",'INDEX.HTM');
$zip->CloseZip();
```

All methods return TRUE on success or FALSE on failure. If the method failed, an (English) error message can be found in \$zip->Error.

- **Package** wascore

Zip::\$central_directory

array = array() [line 145]

- **Var** \$central_directory buffer for the central directory entries

This array is keyed by relative filename (both files and directories), no leading '/' though directories have a trailing '/'.

Zip::\$Error

string = [line 157]

- **Var** \$Error collects error messages if things go wrong

Zip::\$no_name_files

int = 0 [line 169]

- **Var** \$no_name_files is used to construct names for otherwise unnamed files

Zip::\$offset

int = 0 [line 148]

- **Var** \$offset always points to the next local file header in the ZIP-archive

Zip::\$zip_buffer

string = [line 166]

- **Var** \$zip_buffer reference to output buffer if \$zip_type is ZIP_TYPE_BUFFER

Zip::\$zip_comment

string = [line 154]

- **Var** \$zip_comment a file wide comment

Zip::\$zip_filehandle

null|resource = NULL [line 163]

- **Var** \$zip_filehandle handle on the zipfile output if \$zip_type is ZIP_TYPE_FILE

Zip::\$zip_path

string = [line 160]

- **Var** \$zip_path name of the zipfile if \$zip_type is ZIP_TYPE_FILE

Zip::\$zip_type

string = ZIP_TYPE_NONE [line 151]

- **Var** \$zip_type ZIP-archive destination: file, stream or buffer

Constructor *void* function Zip::Zip() [line 172]

constructor initialises all variables

bool function Zip::AddData(\$data, [\$filename = ''], [\$comment = ''], [\$timestamp = 0]) [line 317]

Function Parameters:

- *string* **\$data** the data to add to the ZIP-archive
- *string* **\$filename** the preferred name of the file in the ZIP-archive
- *string* **\$comment** an optional comment for this specific file
- *int* **\$timestamp** the unix timestamp to associate with the file

add data to the current ZIP-archive

This adds the data to the current ZIP-archive.

bool function Zip::AddFile(\$path, [\$filename = "], [\$comment = "]) [*line 276*]

Function Parameters:

- *string* **\$path** the full (absolute) name of the file to add to the ZIP-archive
- *string* **\$filename** the preferred name of the file in the ZIP-archive (default is \$path)
- *string* **\$comment** an optional comment for this specific file

add the contents of an existing file to the current ZIP-archive

this reads the file \$path into a buffer, and subsequently adds the data to the ZIP-archive.

bool function Zip::CloseZip() [*line 344*]

finish the ZIP-archive by outputting the central directory and closing output

this finishes the ZIP-archive by constructing and outputting the Central Directory and subsequently closing the output file (in case of ZIP_TYPE_FILE). The call to CloseZip() is necessary to create a complete ZIP-archive, including the Central Directory.

string function Zip::dos_time_date(\$timestamp) [*line 675*]

Function Parameters:

- *int* **\$timestamp** unix timestamp (seconds since 1970-01-01 00:00:00)

construct an MS-DOS time and date based on unix timestamp

this routine constructs a string of 2 x 2 bytes with the time and the date in the following

format.

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
h h h h h m m m m m m x x x x x
```

hhhhh = hour, from 0 - 23 (5 bits)
mmmmm = minute, from 0 to 59 (6 bits)
xxxxx = duoseconds, from 0 to 29 (5 bits)

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
y y y y y y y m m m m d d d d d
```

yyyyyyy = year offset from 1980, from 0 - 119 (7 bits)
mmmm = month, from 1 to 12 (4 bits)
dddd = day, from 1 to 31 (5 bits)

Note that the time resolution is 2 seconds whereas the unix timestamp has a 1 second resolution. This means that the seconds are rounded down. Also note that the specification [4] indicates that the maximum value for year offset is 119 which corresponds with 2099 rather than the maximum of 127 which would yield the year 2107.

string function Zip::make_suitable_filename(\$filename) [line 622]

Function Parameters:

- *string* **\$filename** name to analyse/massage

construct a suitable filename for use in ZIP-archive

this analyses and edits the string \$filename in such a way that a suitable name for use in a ZIP-archive remains. This means that:

- MS-DOS driverletters are removed from the path
- backslashes are replaced with slashes
- leading './' if any is removed
- a leading slash is removed

bool function Zip::OpenZipbuffer(&\$buffer, [\$comment = ""]) [line 252]

Function Parameters:

- *string* **&\$buffer** a pointer to the buffer where we can write the ZIP-archive
- *string* **\$comment** an optional comment to include in the ZIP-archive

prepare the user supplied buffer for subsequent ZIP-archive data

bool function Zip::OpenZipfile(\$path, [\$comment = "]) [line 195]

Function Parameters:

- *string* **\$path** the (absolute) pathname of the destination file
- *string* **\$comment** an optional comment to include in the ZIP-archive

open a file for subsequent output of ZIP-archive

this opens the file \$path for writing and also sets the zip_type to ZIP_TYPE_FILE. The optional \$comment is stored for future reference. The file must be closed afterwards via [CloseZip\(\)](#).

bool function Zip::OpenZipstream(\$name, [\$comment = "]) [line 223]

Function Parameters:

- *string* **\$name** the name of the ZIP-archive that is suggested to the browser
- *string* **\$comment** an optional comment to include in the ZIP-archive

start with a stream (direct output) indicating an application/zip type of content

this starts the output of the ZIP-archive directly to the browser. The Content-Type and the Content-Disposition are set by sending headers. The stream must be closed afterwards via [CloseZip\(\)](#).

bool function Zip::zip_add_data(&\$data, \$filename, \$comment, \$timestamp) [line 415]

Function Parameters:

- *string* **&\$data** a pointer to a buffer with data to add to the ZIP-archive
- *string* **\$filename** the preferred name of the file in the ZIP-archive
- *string* **\$comment** an optional comment for this specific file (could be "")
- *int* **\$timestamp** the unix timestamp to associate with the file

workhorse function to add data to the current ZIP-archive

This actually adds the data to the current ZIP-archive.

Note that we try to make a wise decision about compressed data: the compressed data should be smaller than the uncompressed data. If not, we don't bother and simply store the data as-is.

We also try to keep the number of copies of the data down to a minimum by not copying the \$data but selecting between \$data and \$zdata only when we are really ready to write output the data.

- **TODO** should we handle the possibility of an additional 4 bytes for DICTID (RFC1950, reference [2])?
- **TODO** should we handle the option of a better compression level (eg. level 9) in gzcompress()? we could check to see if CMF equals 0x78 and FLG is either 0x01, 0x5E, 0x9C or 0xDA the latter 4 values might have an effect on general purpose bit flag bits 2 and 3. for now we'll just keep it simple, but there might be a little something to improve here.

bool function Zip::zip_add_directories(\$pathname, \$timestamp) [*line 519*]

Function Parameters:

- *string* **\$pathname** contains 0, 1 or more directories that lead to the file that needs to be added
- *int* **\$timestamp** unix timestamp to associate with the directory

workhorse function to add 0, 1 or more directories to the current ZIP-archive

this routine works from top to bottom through the specified path, adding directories to the archive. If a particular directory was already added before, it is not added again. This information is based on the existence of the corresponding key in the central_directory array.

void function Zip::zip_error(\$function, \$message) [*line 600*]

Function Parameters:

- *string* **\$function** name of the function/method where the error occurred
- *string* **\$message** the error message to add

add an error message to the list of error messages

Package wasinstall Procedural Elements

install.php

/program/install.php - the main entrypoint for website installation

This is one of the main entry points for Website@School. Other main entry points are /admin.php, /cron.php, /file.php and /index.php. There is also /program/manual.php. Main entry points all define the constant WASENTRY. This is used in various include()ed files to detect break-in attempts.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: install.php,v 1.1.1.1 2011-02-01 13:00:04 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** how prevent third party-access to install.php after initial install? .htaccess? !exists(..../config.php)?
- **TODO** we should make sure that autosession is disabled in php.ini, otherwise was won't work
- **TODO** we should make sure that we can actually set cookies (necessary when logging in).
- **TODO** we should make sure that register globals is off
- **License** [GNU AGPLv3+Additional Terms](#)

```
INSTALL_DIALOG_CANCELLED = 11 [line 52]
INSTALL_DIALOG_CMS = 4 [line 45]
INSTALL_DIALOG_COMPATIBILITY = 6 [line 47]
INSTALL_DIALOG_CONFIRM = 7 [line 48]
INSTALL_DIALOG_DATABASE = 3 [line 44]
INSTALL_DIALOG_DONE = 9 [line 50]
INSTALL_DIALOG_DOWNLOAD = 10 [line 51]
INSTALL_DIALOG_FINISH = 8 [line 49]
INSTALL_DIALOG_INSTALLTYPE = 1 [line 42]
INSTALL_DIALOG_LANGUAGE = 0 [line 41]
INSTALL_DIALOG_LICENSE = 2 [line 43]
```



```
INSTALL_DIALOG_USER = 5 [line 46]  
PROJECT_SITE = websiteatschool.org [line 53]  
WASENTRY = __FILE__ [line 37]
```

Valid entry points define WASENTRY; prevents direct access to include()'s.

```
include_once dirname(__FILE__)." /version.php" [line 57]
```

demodata.php

/program/install/demodata.php - code to install the main demodata

this file, included from /program/install.php, contains a single routine which installs the main demodata. This is done only once during a fresh install of a site.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: demodata.php,v 1.1.1.1 2011-02-01 13:00:57 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function demodata(&\$messages, &\$config) [line 63]

Function Parameters:

- *array* **&\$messages** used to return (error) messages to caller
- *array* **&\$config** pertinent information about the site and also returns the area_id's used for the demo

insert basic demonstration data; the foundation for the module/theme demonstration data

this routine inserts all sorts of demonstration data as a foundation for the demonstration of various modules and themes.

The array &\$messages is used to pass (error) messages back to the caller. The overall result returned is TRUE on success, or FALSE on failure.

The parameter &\$config is used to communicate essential information about the site that is being installed, such as the main URL and the various directories. Also the information about the first user account is passed; this can be used to setup alerts etc. Finally, this array is used to return the three numbers of the three demonstration areas created.

The first demonstration area is a public area. This would be the area to show off all bells and whistles of the CMS. The second demonstration area is a private area. This area could be used to show off an intranet-type of application, maybe accessible only to members of the team. The third area is an in-active area, just for the heck of it.

Note that the demonstration data is to be translated. All translations can be found in /program/install/languages/LL/demodata.php where LL indicates the language code. The language to use is specified in the parameter \$config['language_key'].

This routine is completely self-contained, even the translations are handled manually here.

bool function demodata_alerts(&\$messages, &\$config, &\$tr) [line 882]

Function Parameters:

- *array &\$messages* used to return (error) messages to caller
- *array &\$config* pertinent information about the site
- *array &\$tr* translations of demodata texts

create a few alerts

bool function demodata_areas(&\$messages, &\$config, &\$tr) [line 109]

Function Parameters:

- *array &\$messages* used to return (error) messages to caller
- *array &\$config* pertinent information about the site
- *array &\$tr* translations of demodata texts

create three areas + themes

bool function demodata_sections_pages(&\$messages, &\$config, &\$tr) [line 522]

Function Parameters:

- *array &\$messages* used to return (error) messages to caller
- *array &\$config* pertinent information about the site
- *array &\$tr* translations of demodata texts

create a few sections and pages

bool function demodata_users_groups(&\$messages, &\$config, &\$tr) [line 243]

Function Parameters:

- *array* **&\$messages** used to return (error) messages to caller
- *array* **&\$config** pertinent information about the site
- *array* **&\$tr** translations of demodata texts

create a handful of users/groups/capacities/acls

This routine creates the following 4 groups:

- faculty (principals and teachers)
- team (principals and teachers and all other employees)
- seniors (pupils in grades 5 to 8)
- juniors (pupils in grades 1 to 4)

The following 7 group/capacities are also created

- faculty/principal (3)
- faculty/member (4)
- team/member (4)
- seniors/pupil (1)
- seniors/teacher (2)
- juniors/pupil (1)
- juniors/teacher (2)

The following 8 users are also created

- Amelia Cackle (acackl): Faculty/Principal, Team/Member
- Maria Montessori (mmonte): Faculty/Member, Team/Member, Seniors/Teacher
- Helen Parkhurst (hparkh): Faculty/Member, Team/Member, Juniors/Teacher
- Freddie Frinton (ffrint): Team/Member
- Andrew Reese (andrew): Seniors/Pupil
- Catherine Hayes (catherine): Seniors/Pupil
- Herbert Spencer (herberd): Juniors/Pupil
- Georgina King (georgina): Juniors/Pupil

Every user and every group/capacity gets their own acl

- faculty/principal: access to all private areas
- faculty/member: access to intranet in \$config['private_area_id']
- others get no special privileges

- **TODO** get rid of the \$wizard kludge!
- **TODO** should we append an underscore to the userpaths to make sure we don't clash with the first user account?

index.php

/program/install/index.php - redirector for website installation

The sole purpose of this file is to redirect users from /program/install to /program/install.php. The latter is the main entry point for website installation.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: index.php,v 1.1.1.1 2011-02-01 13:00:59 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

demodata.php

**/program/install/languages/en/demodata.php - translated messages for
/program/install/demodata.php (English)**

This file holds the English texts that are used in the part of the installer that inserts the demonstration data. It is the basis for all other language files.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: demodata.php,v 1.1.1.1 2011-02-01 13:01:00 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

install.php

**/program/install/languages/en/install.php - translated messages for
/program/install.php (English)**

This file holds the English texts that are used in the installer. It is the basis for all other language files.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: install.php,v 1.1.1.1 2011-02-01 13:01:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

demodata.php

**/program/install/languages/nl/demodata.php - translated messages for
/program/install/demodata.php (Dutch)**

This file holds the Dutch texts that are used in the part of the installer that inserts the demonstration data.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: demodata.php,v 1.1.1.1 2011-02-01 13:01:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

install.php

/program/install/languages/nl/install.php - translated messages for /program/install.php
(Dutch)

This file holds the Dutch translations that are used in the installer.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: install.php,v 1.1.1.1 2011-02-01 13:01:02 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

tabledata.php

/program/install/tabledata.php defines core data in a generic way

This file contains the essential data for a new installation, i.e. the items in the configuration table that should exist, etc. This is all done in a generic (database-independent) way. This file defines an array called '\$tabledata' which contains one or more arrays with a tablename and yet another array with fieldname/fieldvalue pairs. This construction with nested arrays is converted to an actual SQL-statement in a function. That function also takes care of the table prefix, so we can simple refer to the bare tablenames here.

The reason to use this nested array construction is that it is easier to see which field gets which value compared to a (possibly very long) SQL-statement. Furthermore you don't need to worry about prefixing the table name and it is almost impossible to mismatch the number of fields and the number of values because they are combined in a \$key => \$value pair. Finally, all strings are automagically escaped with \$DB->escape() in the function that constructs the actual SQL-statement.

This definition file uses the PHP variable types, i.e. if you want to insert a number, you can specify a number (without quotes) and for a boolean you can use the PHP-values TRUE or FALSE. Here's an example. \$tabledata = array();

```
$tabledata[] = array('table' => 'tablename_without_prefix',
    'fields' => array(
        'string_field' => 'This is a string, even with unescaped "quotes"',
        'boolean_field' => TRUE,
        'integer_field' => 123,
        'date_field' => '2008-02-01 23:34:45',
        'double_field' => 1.234567,
        'field_with_null_value' => NULL
    )
);
```

Note that a date field is handled like a string field.

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: tabledata.php,v 1.1.1.1 2011-02-01 13:01:00 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

tabledefs.php

/program/install/tabledefs.php defines all core tables in a generic way

This is the main data definition for Website@School. This file is used by the installation script [install.php](#) to create all main tables.

Here is a reminder for the allowed parameters for field- and key definitions. FIELDS | name | type | len | dec | unsigned* | notnull | default | enum_values | comment |

-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+										
serial*	req	req	-	-	-	-	-	-	-	opt
bool	req	req	-	-	-	opt	opt	-	-	opt
short	req	req	opt	-	opt	opt	opt	-	-	opt
int	req	req	opt	-	opt	opt	opt	-	-	opt
long	req	req	opt	-	opt	opt	opt	-	-	opt
float	req	req	opt	opt	opt	opt	opt	-	-	opt
double	req	req	opt	opt	opt	opt	opt	-	-	opt
decimal	req	req	opt	opt	opt	opt	opt	-	-	opt
number	req	req	opt	opt	opt	opt	opt	-	-	opt
varchar	req	req	opt	-	-	opt	opt	-	-	opt
enum	req	req	opt	-	-	opt	opt	req	-	opt
char	req	req	opt	-	-	opt	opt	-	-	opt
text	req	req	-	-	-	opt	-	-	-	opt
longtext	req	req	-	-	-	opt	-	-	-	opt
blob	req	req	-	-	-	opt	-	-	-	opt
longblob	req	req	-	-	-	opt	-	-	-	opt
date	req	req	-	-	-	opt	opt	-	-	opt
time	req	req	-	-	-	opt	opt	-	-	opt
datetime	req	req	-	-	-	opt	opt	-	-	opt
timestamp	req	req	-	-	-	opt	opt	-	-	opt

INDICES | name | type | unique | fields | reftable | reffields | comment |

-----+-----+-----+-----+-----+-----+-----+						
primary	-	req	-	req	-	opt
index	opt	req	opt	req	-	opt
foreign	opt	req	-	req	req	opt

req = required, opt = optional, - = not allowed

- **Package** wasinstall
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: tabledefs.php,v 1.1.1.1 2011-02-01 13:00:59 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker

- **TODO** automatically create appropriate sequence name for serial fields??? or add seqdefs too?
- **License** [GNU AGPLv3+Additional Terms](#)

Package wasinstall Classes

Class InstallWizard

[line 109]

class for performing installation tasks

Overview

Dialog screens

The installer basically consists of some six dialog screens where the user is supposed to enter some data (e.g. language, install type, etc.). Each of those screens has a [Next] button and most screens have a [Previous] button. Also, every screen has a [Cancel] button. The [Cancel] button always immediately leads to the Cancel screen and the whole process is stopped (by resetting the collected information in `$_SESSION['INSTALL']`). The [Next] button always validates/processes the data the user entered. If the results are good, we go to the next step. If the processing fails, we stay where we are (ie. the current dialog is re-displayed). The [Previous] button backs up one step, without saving or storing the user entered data; the user **MUST** press [Next] to save the data entered.

The Finish-dialog

The FINISH-dialog has no [Previous] button, because all the real work is done when the user presses [Next] in the CONFIRM-dialog. This is a one-time action (creating tables, filling with demodata, etc.) so it makes no sense to backup at that point.

The stage variable and backing up via the menu

The variable 'stage' moves along with the highest dialog the user has successfully reached. This variable is responsible for greying out/disabling the menu options. The menu can be used to jump back a few steps in the procedure. However, once the transition from CONFIRM to FINISH is made, it is no longer possible to return to previous steps (it makes no sense to do so because at that point the real work is already done). The jump to a particular step/dialog is done via the GET-parameter 'step'. The buttons all work via the POST'ed parameter dialog.

Special cases

There are a few special cases:

- download: this yields an immediate download of the constructed config.php and no further dialog is displayed
- done: this is a pseudo-dialog. In effect it is a redirect to the newly created site (either admin.php or index.php or perhaps manual.php).

- **Package** wasinstall

InstallWizard::\$license

string = [line 117]

- **Var** \$license ready-to-use HTML-code with the text of the license from /program/license.html

InstallWizard::\$messages

string = array() [line 111]

- **Var** \$messages collects error messages if any

InstallWizard::\$results

array = array() [line 114]

- **Var** \$results collects outcome of various compatibility results in human readable form

Constructor *void* function InstallWizard::InstallWizard() [line 125]
constructor

this constructs the install wizard and also makes sure that the INSTALL-array (kept in the \$_SESSION array) is initialised with default values if it did not already exist.

string function InstallWizard::appropriate_legal_notices(\$high_visibility, [\$m = ""]) [*line 2425*]

Function Parameters:

- *bool* **\$high_visibility** if TRUE we return a text-only link, otherwise a clickable image
- *string* **\$m** margin to improve readability of generated code

construct a link to appropriate legal notices as per AGPLv3 section 5

This routine constructs ready-to-use HTML-code for a link to the Appropriate Legal Notices, which are to be found in /program/about.html. Depending on the highvisibility flag we either generate a text-based link or a clickabel image.

The actual text / image to use depends on the global constant WAS_ORIGINAL. This constant is defined in /program/version.php and it should be TRUE for the original version of Website@School and FALSE for modified versions.

In the former case the anchor looks like 'Powered by Website@School', in the latter case it will look like 'Based on Website@School', which is in line with the requirements from the license agreement for Website@School, see /program/license.html.

IMPORTANT NOTE

Please respect the license agreement and change the definition of WAS_ORIGINAL to FALSE if you modify this program (see /program/version.php). You also should change the file '/program/about.html' and add a 'prominent notice' of your modifications.

Note: a comparable routine can be found in [waslib.php](#).

string function InstallWizard::button(\$button) [*line 2693*]

Function Parameters:

- *string* **\$button** indicates which button to create, e.g. 'next', 'previous', 'cancel', 'finish', 'ok'.

shorthand for creating a submit button in the correct style

bool function InstallWizard::check_compatibility() [*line 1263*]

check certain compatibility issues and optionally return test results

this routine performs a few tests and returns an overall go/nogo signal Human readable test results are stored in \$this->results. Return TRUE on passing all tests, FALSE otherwise + errors in \$this->messages

- **TODO** add more tests, e.g. for gd, safe_mode, memory limit, etc.

bool function InstallWizard::check_for_nameclash(\$demodata, \$label, \$username) [line 3032]

Function Parameters:

- *bool* **\$demodata** is installation of demodata requested?
- *string* **\$label** name of the username field in dialog
- *string* **\$username** the proposed username for the webmaster account

check for name clash of new user (webmaster) and user accounts from demodata

This routine checks to see if the name the webmaster supplied is not one of the demodata accounts. If so, we flag the error in the messages.

Note: The list of accounts must be updated whenever the demodata is updated. This is a kludge but I'll leave it this way for the time being. See also the main file [demodata.php](#).

bool function InstallWizard::check_license() [line 474]

check if the user accepts the licences

This is the companion routine for [show_dialog_license\(\)](#). It checks whether the user dutifully type 'I agree'.

bool function InstallWizard::check_validation() [line 1497]

shorthand to check the validation status of the relevant dialogs

this checks the various validation flags. If a flag is false, the corresponding error message is added to \$this->messages and the function returns FALSE.

void function InstallWizard::clamscan_installed(&\$clamscan_path, &\$clamscan_version) [line 3295]

Function Parameters:

- *string* **&\$clamsan_path** path to binary program (output)
- *string* **&\$clamsan_version** version of the program we found (output)

try to locate clamdscan or clamscan on the server

This routine checks to see if either clamdscan or clamscan can be found somewhere. This is done via educated guessing. On success we return the path to the binary program file in \$clamsan_path and the output of the version command in \$clamsan_version. The function returns TRUE if we did find the clamscan program.

Note that we scan the directories with opendir/readdir/closedir rather than rely on file_exists() etc. because we would not find binaries with permissions 0711, which would otherwise be perfectly acceptable to execute with exec().

string function InstallWizard::construct_config_php() [line 2228]

prepare a configuration file based on the collected information

bool function InstallWizard::create_tables(\$filename) [line 3172]

Function Parameters:

- *string* **\$filename** contains the table definitions

create tables in database via include()'ing a file with tabledefs

- **Uses \$DB**

void function InstallWizard::end_session_and_redirect() [line 2054]

unset installation data, end session and redirect the user to elsewhere

this redirects the user to one of the possible destinations as selected in the finish dialog (see [show dialog finish\(\)](#)). Also, the INSTALL-array is unset, effectively erasing the collected data from the session. Subsequently the session cookie is reset, effectively ending the session. The effect is that the user has to start over if she returns to install.php. (The equivalent of 'logging off').

void function InstallWizard::errorcount_bump() [*line 2907*]

increment the error counter and perhaps slow things down

this routine counts the number of errors. It is used to count the number of attempts to guess a valid host/username/passord triplet. If the number of errors reaches 3, a delay is added (the installation program sleeps for a while). On every additional error an extra delay is added (3 seconds at a time) until the total delay reaches 24 seconds. At that point the collected data are reset and effectively the user has to start over.

void function InstallWizard::errorcount_reset() [*line 2924*]

reset the error counter

this routine resets the effect of [errorcount_bump\(\)](#).

int function InstallWizard::fetch_license(&\$license) [*line 2748*]

Function Parameters:

- *string* **&\$license** receives ready-to-use HTML-code with the text of the license from /program/license.html

helper to retrieve the text of the LICENSE AGREEMENT for Website

bool function InstallWizard::gd_supported(&\$details) [*line 3365*]

Function Parameters:

- *string* **&\$details** returns detailed information about GD version and supported file formats

retrieve information about GD and supported graphics file formats

this routine determines whether GD is enabled and which graphics file formats are supported (check for GIF, JPG and PNG). If GD is not installed or if none of these three formats are supported, the routine returns FALSE. Details (version number, supported formats) are returned in &\$details.

GIF is a special case due to patent issues: there is a distinction between read support and write support (see the PHP-documentation for details). GD-version >= 2.0.28 provides full support. This routine makes the distinction between fully supported (GIF: Yes), reading but not writing (GIF: Readonly) and not supported (GIF: No).

array function InstallWizard::get_default_install_values() [*line 2094*]

return an array with default configuration values

this routine tries to calculate/guess the best default values for config.php. We do so by looking in the global `$_SERVER` variable. If that doesn't work, we simply make up sensible values. In the end it is up to the user to enter the correct data; the values here are mere defaults.

A first-time install should be possible without changing/editing the default values, i.e. a standard Next-Next-Finish-type of installation.

- **TODO** should we check the program version versus the stored program version here?
- **TODO** there is something wrong with the default for `$cms_www`; FIXME (commented out for now)

array function InstallWizard::get_dialogdef_cms() [line 964]

fill an array with necessary information for the cms dialog

Note that this is a very light-weight implentation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

Note: the order of 'cms_demodata' and 'cms_demodata_password' is important: the field 'cms_demodata' must come first. If not, the validation of the password is not skipped if demodata is left unchecked by the user. See also [save cms\(\)](#).

array function InstallWizard::get_dialogdef_database() [line 678]

fill an array with necessary information for the database dialog

Note that this is a very light-weight implentation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

array function InstallWizard::get_dialogdef_finish() [line 1988]

fill an array with necessary information for finish / jump dialog

Note that this is a very light-weight implentation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

array function InstallWizard::get_dialogdef_installtype() [line 388]

fill an array with necessary information for installtype dialog

Note that this is a very light-weight implementation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

array function InstallWizard::get_dialogdef_language() [line 299]

fill an array with necessary information for language dialog

Note that this is a very light-weight implementation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

array function InstallWizard::get_dialogdef_user() [line 1158]

fill an array with necessary information for the first user dialog

Note that this is a very light-weight implementation of the dialogdef idea used in the main program: we don't do fancy stuff with labels, hotkeys, etc. KISS, because I don't want to rely on all the libraries of the main program with all their interconnections and dependencies; the installer should more or less be a stand-alone application.

array function InstallWizard::get_list_of_install_languages() [line 2714]

retrieve a list of available languages by querying the file system for install.php translation files

this routine constructs a list of language codes and language names (in the languages themselves) based on the language subdirectories available under /program/install/. The resulting array of code-name-pairs is sorted by name.

Note that because the names of the languages are expressed in the languages themselves, this routine has the side-effect of reading all of the available language files into memory (see [t\(\)](#)).

array function InstallWizard::get_manifests(\$path) [line 3147]

Function Parameters:

- *string \$path* top directory for the search for manifest files

retrieve an array of manifests for modules, themes or languages

this examines the file system starting in the directory \$path, looking for manifest files.

These manifest files are named after the subdirectory they are in as follows. Example: If \$path is /program/modules, this routine steps through that directory and may find subdirectories 'htmlpage', 'guestbook' and 'forum'. Eventually these manifest files are include()'d:

/program/modules/htmlpage/htmlpage_manifest.php,	
/program/modules/guestbook/guestbook_manifest.php	and
/program/modules/forum/forum_manifest.php.	

Every manifest file must describe the module (or language or theme) via the following construct:

```
$manifests['htmlpage'] = array('name' => 'htmlpage', ..., 'cron_interval' => 0);
```

After processing all the subdirectories of \$path, the resulting array \$manifests is returned. Note that pseudo-directories like '.' and '..' are not considered. Also, subdirectories 'foo' without the file 'foo_manifest.php' are also ignored.

Note that the name of the manifest file itself is also stored in the array, but excluding the subdirectory name.

string function InstallWizard::get_menu(\$dialog, \$stage, [\$m = ""]) [line 2484]

Function Parameters:

- *int* **\$dialog** indicates the current dialog
- *int* **\$stage** indicates the highest numbered dialog that was already reached
- *string* **\$m** margin for better readability of generated HTML-code

construct a clickable menu which helps the user to jump back and forth in the funnel

this constructs a menu that allows the user to jump to a another step in the procedure when appropriate. Two parameters are important: the \$dialog and the the \$stage. The \$dialog indicates the current dialog, i.e. the dialog that is currently displayed in the content area. This item in the menu is emphasised (e.g. the link is underlined via the style sheet). The \$stage indicates how far we are in the procedure. The installation consists of some eight steps, and the used is encouraged to perform all steps in the natural order, by repeatedly pressing the [Next] button in the dialogs. Every time a dialog appears to have valid data, the \$stage is incremented.

All the menu items after the current stage are greyed out and basically inaccessible for the user. Menu items before the current stage are accessible, so it is possible to jump backwards to dialogs that were already processed but the only way to advance to a new screen is to use the [Next] (and provide valid data, obviously).

The greyed-out menu items have a href property consisting of a simple "#". This is interpreted by the browser as a relative link within the current page. The effect is that the current page stays on the screen, including any unsaved data the user may already have entered. By showing the links in a different colour (grey and not blue), the user can visually see which items are clickable and which are not.

By showing all the menu items and greying-out the inaccessible ones we effectively build a funnel and at the same time indicating which steps will follow in the procedure.

There is a special case when \$stage hits the FINISH-dialog. If the user has not yet reached that stage, all dialogs before the finish-dialog are active and the rest is greyed-out. Once the \$stage reaches the FINISH-dialog, all previous dialogs become instantly unreachable. This is because the step between the confirmation dialog and the finish dialog is the place where the actual installation takes place. That is a one-time operation and the user should not be able to jump backwards and change data after all the actual installation work is already done.

Note that the menu item to download the config.php is displayed before the finish dialog. This is because it appears more logical to me (but YMMV).

array function InstallWizard::get_options_db_type() [line 2888]

construct a list of database options

This constructs an array with key-value-pairs indicating all available databases. Currently (september 2009) there is only one database supported: mysql. However, in the future we may support more databases, such as PostgreSQL.

string function InstallWizard::get_page([\$dialog_title = "], [\$menu = "], [\$content = "], [\$help_topic = 'install']) [line 2279]

Function Parameters:

- *string \$dialog_title* text to show in the browser's title bar (indicating where we are)
- *string \$menu* ready-to-use HTML-code comprising the menu at the left hand side of the page
- *string \$content* ready-to-user HTML-code holding the actual contents of the page
- *string \$help_topic* the topic or subtopic in the manual to link to

construct a complete HTML-page that can be sent to the user's browser

this routine constructs a string containing the complete page to send to the user's browser, starting with the <html> opening tag and ending with the </html> closing tag. The constructed page is returned as a string.

This routine also peeks into the INSTALL-array, e.g. for the language key and the high_visibility flag.

If \$this->messages is not empty, the items in that array are displayed between the page header (logo+helpbutton) and the menu/content area. This is the feedback of the previous action to the user (if any).

- **TODO** should we promote language and high_visibility to function parameters instead of using \$_SESSION directly?

array function InstallWizard::guess_url() [line 2576]

educated guesses for scheme, host and portname from \$_SERVER

this routine tries to guess the various components of the url that was used to reach this script, based on the information in the global array \$_SERVER. If no information can be guessed at all, the result is something like 'http://localhost'.

Note that the 'authority' is a combination of hostname and portnumber, but only if the portnumber is non-standard. For http and port 80, and https and port 443 the portnumber is suppressed, because these are the default ports for those schemes.

Note that we actually guess the url that should correspond with the document root.

bool function InstallWizard::insert_tabledata(\$filename) [line 3204]

Function Parameters:

- *string \$filename* contains the table definitions

fill tables in database via include()'ing a file with tabledata

- **Uses \$DB**

void function InstallWizard::is_already_installed() [line 3052]

check for previous install

this routine checks to see if another installation should be allowed. Returns TRUE if the program was already installed or FALSE otherwise.

string function InstallWizard::magic_unquote(\$value) [line 2785]

Function Parameters:

- *string* **\$value** a string value that is conditionally unescaped

this circumvents the 'magic' in `magic_quotes_gpc()` by conditionally stripping slashes
 This routine borrowed from [waslib.php](#).

bool function InstallWizard::perform_installation() [*line 1557*]

perform the actual initialisation of the cms

this routine initialises the database: creates tables, inserts essential data (first user account, other defaults) and optional demonstration data.

The strategy is as follows.

- (1) manufacture a database object in the global \$DB
- (2A) create the main tables (from /program/install/tabledefs.php)
- (2B) insert essential data (from /program/install/tabledata.php)
- (2C) store the collected data (website title, etc.),
- (2D) create the first useraccount,
- (3) if necessary, create the data directory
- (4) record the currently available languages in the database

Once the main part is done, install modules and themes based on the relevant information that is stored in the corresponding manifest-file by performing the following steps for each module and theme:

- (5A) insert a record in the appropriate table with active = FALSE
- (5B) create the tables (if any tables are necessary according to the manifest)
- (5C) install the item by including a file and executing function <item>_install()
- (5D) flip the active flag in the record from step 5A to indicate success
 Subsequently the optional demodata is installed.
- (6A) a foundation is created via the function demodata() from /program/install/demodata.php
- (6B) all modules + themes can add to the demo data via the appropriate subroutines
 If all goes well, this routine ends with an attempt to
- (7) save the config.php file at the correct location.
 (it is not an error if that does not work; it only means that the user has to upload the config.php file manually.

- **TODO** should we save the config.php to the datadir if the main dir fails? Mmmm.... security implications?
- **TODO** this routine badly needs refactoring

void function InstallWizard::quasi_random_string(\$length, [\$candidates = 36]) [*line 3236*]

Function Parameters:

- *int* **\$length** length of the string to generate
- *int* **\$candidates** number of candidate-characters to choose from

generate a string with quasi-random characters

This routine borrowed from [waslib.php](#).

string function InstallWizard::render_dialog(\$dialogdef, \$m) [*line 2818*]

Function Parameters:

- *array* **\$dialogdef** contains labels, values and other information describing input elements
- *string* **\$m** improves readability of generated code

quick and dirty dialogdef renderer

This is a small routine to render simple dialogs with strings, lists and checkboxes.

Note that every element in the \$dialogdef is in itself an array. Recognised elements in those arrays are:

- label (string): displayed before the actual input element
- help (string): additional information for the user, rendered after/under the input element
- value (mixed): the current value of the input element
- show (bool): if TRUE, the element is displayed/rendered, otherwise it is simply skipped
- type (enum): 's'=>string (text), 'p'=>password, 'l'=>list, 'b'=>bool(checkbox)
- options(array): array with key-value-pairs with acceptable choices (used in 'l' (list) elements)
- minlength(int): minimum length of an input string or password
- maxlength(int): maximum length of an input string or password

The names of the input elements are copied from the keys used in \$dialogdef.

void function InstallWizard::run() [*line 140*]

main dispatcher for the Installation Wizard

This routine termines what needs to be done and does it by calling the corresponding workhorse routines.

string function InstallWizard::sanitise_filename(\$filename) [*line 3254*]

Function Parameters:

- *string* **\$filename** the string to sanitise

sanitise a string to make it acceptable as a filename/directoryname

This routine borrowed from [waslib.php](#).

bool function InstallWizard::save_cms() [*line 796*]

validate and store the CMS-data the user supplied

This is the companion routine for [show_dialog_cms\(\)](#). It stores the user-supplied data about the website (paths etc.) We always store the data in the global `_SESSION`, even if something goes wrong. This makes that the user will use the latest values when re-doing the dialog.

We try to validate the specified directories: they should at least exist. The datadirectory should also be writable by us. If it not exists we try to create it (and remove it after the test). Note that the PHP `safe_mode` may complicate things here.

- **TODO** also take `safe_mode` into account? Should that be a requirement for succesfull installation?

bool function InstallWizard::save_database() [*line 564*]

validate database information

This is the companion routine for [show_dialog_database\(\)](#). It stores the user-supplied data about the database. We always store the data in the global `_SESSION`, even if something goes wrong. This makes that the user will use the latest values when re-doing the dialog. However, only when the values are valid, the parameter `db_validated` is set to `TRUE`. This is used lateron (in the confirmation phase).

This routine doubles as a gate keeper. Every time the user makes a mistake, an error counter is incremented and the script pauses for some time (see [errorcount_bump\(\)](#)). If there are too many errors, the script resets the data collected sofar and the procedure starts from scratch. This is a (probably futile) attempt to make it harder to brute force an entry by repeatedly

probing for database credentials. Unfortunately there is no easy way (at least one I can think of) to protect this script from repeated break-in attempts other than by simply removing or renaming this script. Oh well.

The following tests are performed:

- fields should not fail basic tests (min/max stringlength etc.)
- the database is not supposed to be in the list of 'forbidden' databases (e.g. 'test' or 'mysql')
- prefix must only use letters, digits or an underscore
- prefix must start with a letter

If the above conditions are not satisfied we bail out immediately, without testing other information. Otherwise, we also perform these tests:

- is it possible to connect to the database server
- is it possible to select the specified database
- are there no tables in the database that start with the prefix

If something needs to be done about the prefix and we are in standard mode, we automatically switch to custom mode, allowing the user to edit the prefix too.

bool function InstallWizard::save_installtype() [line 355]

store the selected install type + high visibility flag

This is the companion routine for [show_dialog_installtype\(\)](#). It stores the user-supplied choices for custom install and high visibility. The values in the radio button should be 0 for standard and 1 for custom install. The value we keep is a boolean indicating a custom install yes or no.

bool function InstallWizard::save_language() [line 269]

store the selected language

This is the companion routine for [show_dialog_language\(\)](#). It validates and stores the user-supplied language key.

bool function InstallWizard::save_user() [line 1106]

validate and store the data for the first user account

This is the companion routine for [show_dialog_user\(\)](#). It stores the information about the first user account. We always store the data in the global `_SESSION`, even if something goes wrong. This makes that the user will use the latest values when re-doing the dialog.

We try to validate at least the password:

- minimum of 8 characters
 - minimum 1 upper case, 1 lower case, 1 digit
- Also, both username and full name should not be empty

void function InstallWizard::show_dialog_cancelled([\$m = ""]) [line 2020]

Function Parameters:

- *string* **\$m** margin for better readability of generated HTML-code

show the user that the process has been cancelled

this shows a screen with the message that the installation procedure has been cancelled. There is a single [OK] button that effectively allows the user to try again. The existing session (and all the data it might contain) is destroyed.

Note that we first construct the page (possibly in another language) in a separate variable before we destroy all data collected so far.

void function InstallWizard::show_dialog_cms([\$m = ""]) [line 757]

Function Parameters:

- *string* **\$m** margin for better readability of generated HTML-code

construct the dialog for essential cms data (title, paths, e-mail address)

This dialog contains the following fields:

- website_title (varchar(255))
- website_from_address (varchar(255))
- website_replyto_address (varchar(255))
- cms_dir (varchar(240))
- cms_www (varchar(240))
- cms_prokdir (varchar(240))
- cms_progwww (varchar(240))
- cms_dataroot (varchar(240))
- cms_demodata (boolean)
- cms_demodata_password (varchar(255) (but only a sha1() or md5() hash is stored eventually)

Some fields are suppressed in the dialog if the user selected a Standard installation:

- website_replyto_address: copied from website_from_address
- cms_prokdir: constructed from cms_dir
- cms_progwww: constructed from cms_www

- **TODO** can we suppress even more fields here in case of a Standard installation?

void function InstallWizard::show_dialog_compatibility([\$m = ""]) [line 1211]

Function Parameters:

- *string \$m* margin for better readability of generated HTML-code

construct the compatibility overview

this routine displays a tabular overview of minimal compatibility requirements and the current status/testresults. The table is constructed in a subroutine; we only deal with the display here.

Q: Why here, at the last stop before installation? A: Because we otherwise would leak information about our environment to complete strangers (assuming anyone can execute this script).

- **TODO** more tests to perform here: safe mode, memory limit, processing time limit, register globals

void function InstallWizard::show_dialog_confirm([\$m = ""]) [line 1418]

Function Parameters:

- *string \$m* margin for better readability of generated HTML-code

construct the overview/confirmation dialog

This dialog contains an overview of the information entered (excluding the passwords which are visually replaced with asterisks). This is the last chance the user gets to change the data entered. Once the user presses the [Next] button, the actual installation takes off.

void function InstallWizard::show_dialog_database([\$m = ""]) [line 506]

Function Parameters:

- *string \$m* margin for better readability of generated HTML-code

construct the dialog for database (server, host, username, password, etc.)

This dialog contains the following fields:

- db_type (pick from enumerated list)
- db_server (varchar(240))
- db_username (varchar(240))
- db_password (varchar(240))
- db_name (varchar(240))
- db_prefix (varchar(240))

One field is suppressed in the dialog if the user selected a Standard installation:

- db_prefix

void function InstallWizard::show_dialog_finish([\$m = ""]) [line 1934]

Function Parameters:

- *string \$m* margin for better readability of generated HTML-code

construct the finish screen

this dialog is displayed after a succesful installation. The user is prompted to select the next destination, which can be either admin.php (the backoffice), index.php (the frontpage), manual.php (the documentation) or the project's home page.

There is also an option to download config.php. Once the user's choice is submitted, the session is reset and config.php can no longer be downloaded. This session reset is done in the 'Done' dialog (see [end session and redirect\(\)](#)).

Note that the boolean parameter `INSTALL['config_php_written']` indicates whether the file config.php was already succesfully written in the designated location. If this is the case, we show a different message compared to the case where the user still needs to download+upload config.php.

void function InstallWizard::show_dialog_installtype([\$m = ""]) [line 323]

Function Parameters:

- *string \$m* margin for better readability of generated HTML-code

construct the installtype + high visibility selection dialog

This dialog contains a radio button where the user selects 'standard' or 'custom' and also a checkbox for high visibility. As always the dialog ends with buttons to move forward, backward or to cancel the installation process altogether.

`void function InstallWizard::show_dialog_language([$m = ""]) [line 240]`

Function Parameters:

- *string* **\$m** margin for better readability of generated HTML-code

construct the language selection dialog

this dialog allows the user to pick a language. The choices are determined by looking for translation files in the file system, specifically for files `/program/install/languages/LL/install.php` where LL is the language code, see [get list of install languages\(\)](#).

`void function InstallWizard::show_dialog_license([$m = ""]) [line 433]`

Function Parameters:

- *string* **\$m** margin for better readability of generated HTML-code

construct a full license agreement and an input where the user must enter 'I agree'

This constructs a (long) license agreement dialog. We more or less force the user to actually scroll through it by having the input box + buttons after the agreement text. Note that the phrase to enter into the box is also translated, it may be 'I agree' in English but something else in other languages (done via translation file). If the user already accepted the license, the 'I agree'-text is already displayed in the dialog, in the current language. (A user could type 'I agree' in English, change the language into Dutch, in which case the \$value in the textbox would be something like 'Ik ga accoord' or whatever the translation of 'I agree' is.) This is done via an extra level of translation with the 'dialog_license_i_agree' translation in the prompt.

Also, the instruction to enter the exact words are repeated near the bottom of the screen, just to make sure the user understands what to do.

Note that the phrase the user enters is compared to the requested phrase in a case-INsensitive way (see [check_license\(\)](#)).

`void function InstallWizard::show_dialog_user([$m = ""]) [line 1065]`

Function Parameters:

- *string* **\$m** margin for better readability of generated HTML-code

construct the dialog for the first user account

This dialog contains the following fields:

- `user_username` (varchar(255))
- `user_full_name` (varchar(255))
- `user_email` (varchar(255))
- `user_password` (varchar(255) (but only a sha1() or md5() hash is stored eventually)

An additional feature of this routine is to set a default email address for the account, by copying the address that can be found in the Reply-To: field that was entered earlier (or constructed from the From: address).

string function InstallWizard::t(\$key, [\$replace = "], [\$language = "]) [*line 2649*]

Function Parameters:

- *string* **\$key** the key in the string-array with translations
- *array* **\$replace** contains key-value pairs that are used to search/replace in the translated string
- *string* **\$language** indicates which language we should translate into

retrieve a translated string with optional parameters filled in

this routine tries to find a translated string based on the \$key. If \$replace is not empty, any keys in that array that are found in the translation are replace by the corresponding value.

The translations are read from the files /program/install/languages/LL/install.php, where LL is a two-letter language code (e.g. en, nl, de, fr). If the desired language is not available, English is used instead. If the requested translation is not found, the key of the translation is returned, sandwiched between the language codes. Usually this does not happen (all keys used have a translation), but it can be helpful when creating and testing a new translation.

Note that a language file is retrieved completely. This means that all the translations are read from file in one swoop; there is no need to go to the disk for every individual translation. The translation file is buffered in memory via the static variable \$phrases, which is an array keyed by language. That means that multiple languages can co-exist in this static variable. That last feature is used in constructing a list of available languages where the name of the language is expressed in the language itself (see [get_list_of_install_languages\(\)](#)).

Note: By convention the keys in \$replace are upper case words, with optional underscores, sandwiched between curly braces. Examples: '{FIELD}', '{DATABASE}', '{RELEASE_DATE}'. The idea is that these words make life easier for translators.

TRUE function InstallWizard::validate(\$item, \$value) [*line 2941*]

Function Parameters:

- *array* **\$item** this array holds a field definition from a dialog definition

- *string* **\$value** this is the `magic_unquote()`'d and `trim()`'ed value the user POSTed

minimal validation of data input

this routine is a KISS-validator; we check for min/max stringlengths in strings and passwords (defaults: 0 and 255) and a valid item in listboxes. In case of error, a message is added to `$this->messages` and the function returns FALSE. On success we return TRUE. Note that additional validation could or should be done on some fields, e.g. a password of sufficient complexity, etc. This routine does not do that.

TRUE function `InstallWizard::validate_password($label, $password, [$min_lower = 1], [$min_upper = 1], [$min_digit = 1])` [line 2987]

Function Parameters:

- *string* **\$label** this string holds the human-readable field name
- *string* **\$password** this is the `magic_unquote()`'d and `trim()`'ed password the user POSTed
- *int* **\$min_lower** the minimum number of lower case letters
- *int* **\$min_upper** the minimum number of upper case letters
- *int* **\$min_digit** the minimum number of digits

validation of password input

this routine analyses the password provided against the minimal requirements of password complexity.

bool function `InstallWizard::write_config_php()` [line 3089]

attempt to write the file config.php in the correct location

this routine tries to write the file `config.php`. If this fails, we return FALSE, otherwise TRUE. Note that the permissions of the file are set to read-only (`chmod 0400`) for the owner of the file, and nothing for group and world. That should be enough for the webserver.

- **TODO** should we make the filemode (hardcoded at 0400) configurable/customisable?

Package waslang_en Procedural Elements

en_manifest.php

/program/languages/en/en_manifest.php - description of the main language/translation (English)

This file defines the English language package ('en'). This file is used when this essential (core) package is installed.

- **Package** waslang_en
- **Author** Peter Fokker < peter@berestijn.nl >o
- **Version** \$Id: en_manifest.php,v 1.1.1.1 2011-02-01 12:59:54 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package waslang_nl Procedural Elements

nl_manifest.php

/program/languages/nl/nl_manifest.php - description of the Dutch translation package

This file defines the Dutch language package('nl'). This file is used when this translation is installed.

- **Package** waslang_nl
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: nl_manifest.php,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package wasmod_guestbook Procedural Elements

guestbook_admin.php

/program/modules/guestbook/guestbook_admin.php - management interface for module

This file defines the administrative interface to this module. The interface consists of the following four functions.

```
guestbook_disconnect(&$output,$area_id,$node_id,$module)
guestbook_connect(&$output,$area_id,$node_id,$module)
guestbook_show_edit(&$output,$area_id,$node_id,$module,$viewonly,$edit_again,$href)
guestbook_save(&$output,$area_id,$node_id,$module,$viewonly,&$edit_again)
```

These functions are called from pagemanagerlib.php whenever necessary.

- **Package** wasmod_guestbook
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: guestbook_admin.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function guestbook_connect(&\$output, \$area_id, \$node_id, \$module) [*line 71*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which we need to connect

- *array* **\$module** the module record straight from the database

connect this module to a node

this makes the link between the node \$node_id in area \$area_id and this module. It depends on the module what this function should do. Often it simply boils down to a no-op returning TRUE to indicate success.

bool function guestbook_disconnect(&\$output, \$area_id, \$node_id, \$module) [*line 52*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node from which we need to disconnect
- *array* **\$module** the module record straight from the database

disconnect this module from a node

this breaks the link between the node \$node_id in area \$area_id and this module. It depends on the module what this function should do. Often it simply boils down to a no-op returning TRUE to indicate success.

bool function guestbook_save(&\$output, \$area_id, \$node_id, \$module, \$viewonly, &\$edit_again, \$edit_again) [*line 153*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which we need to connect
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing and hence saving is not allowed
- *bool* **\$edit_again** set to TRUE if we need to edit the content again, FALSE otherwise
- **&\$edit_again**

save the modified content data of this module linked to node \$node_id

this validates and saves the data that was submitted by the user. If validation fails, or storing the data doesn't work, the flag \$edit_again is set to TRUE and the return value is FALSE.

If the user has cancelled the operation, the flag \$edit_again is set to FALSE and the return value is also FALSE.

If the modified data is stored successfully, the return value is TRUE (and the value of \$edit_again is a don't care).

Here is a summary of return values.

- `retval = TRUE` ==> data saved successfully
- `retval = FALSE && edit_again = TRUE` ==> re-edit the data, show the edit dialog again
- `retval = FALSE && edit_again = FALSE` ==> cancelled, do nothing

bool function guestbook_show_edit(&\$output, \$area_id, \$node_id, \$module, \$viewonly, \$edit_again, \$href) [*line 104*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which this module is connected
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing is not allowed (but simply showing the content is allowed)
- *bool* **\$edit_again** if TRUE start with data from \$_POST, else use data from database
- *string* **\$href** the action property of the HTML-form, the place where data will be POST'ed

present the user with a dialog to modify the content that is connected to node \$node_id

this prepares a dialog for the user filled with existing data (if any), possibly allowing the user to modify the content. If the flag \$viewonly is TRUE, this routine should only display the content rather than let the user edit it. If the flag \$edit_again is TRUE, the routine should use the data available in the \$_POST array, otherwise it should read the data from the database (or wherever the data comes from). The parameter \$href is the place where the form should be POST'ed.

The dialog should be added to the \$output object. Useful routines are:

```
$output->add_content($content): add $content to the content area  
$output->add_message($message): add $message to the message area (feedback to the user)  
$output->add_popup_bottom($message): make $message popup in the browser after loading the page (uses  
javascript)  
$output->add_popup_top($message): make $message popup in the browser before loading the page (uses  
javascript)
```

guestbook.php

/program/modules/guestbook/languages/en/guestbook.php - translated messages for module (English)

- **Package** wasmod_guestbook
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: guestbook.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

guestbook.php

/program/modules/guestbook/languages/nl/guestbook.php - translated messages for module (Dutch)

- **Package** wasmod_guestbook
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: guestbook.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package wasmod_htmlpage Procedural Elements

htmlpage_admin.php

/program/modules/htmlpage/htmlpage_admin.php - management interface for **htmlpage-module**

This file defines the administrative interface to this module. The interface consists of the following four functions.

```
htmlpage_disconnect(&$output,$area_id,$node_id,$module)
htmlpage_connect(&$output,$area_id,$node_id,$module)
htmlpage_show_edit(&$output,$area_id,$node_id,$module,$viewonly,$edit_again,$href)
htmlpage_save(&$output,$area_id,$node_id,$module,$viewonly,&$edit_again)
```

These functions are called from pagemanagerlib.php whenever necessary.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_admin.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function htmlpage_connect(&\$output, \$area_id, \$node_id, \$module) [*line 73*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which we need to connect

- *array* **\$module** the module record straight from the database

connect this module to a node

this makes the link between the node \$node_id in area \$area_id and this module. In this case we simply link a data container to node \$node_id in a 1-to-1 relation. Note that we might decide later on to keep versions of pages around, e.g. by inserting new records every save rather than updating the existing.

bool function `htmlpage_disconnect(&$output, $area_id, $node_id, $module)` [line 53]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node from which we need to disconnect
- *array* **\$module** the module record straight from the database

disconnect this module from a node

this breaks the link between the node \$node_id in area \$area_id and this module. For now we simply delete the record with page data. In a future version we might want to retain the page data, 'for future reference' (but: what do we do with it? Oh well).

bool function `htmlpage_save(&$output, $area_id, $node_id, $module, $viewonly, &$edit_again)` [line 180]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which the content is connected
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing and hence saving is not allowed
- *bool* **&\$edit_again** set to TRUE if we need to edit the content again, FALSE otherwise

save the modified content data of this module linked to node \$node_id

this validates and saves the data that was submitted by the user. If validation fails, or storing the data doesn't work, the flag \$edit_again is set to TRUE and the return value is FALSE.

If the user has cancelled the operation, the flag \$edit_again is set to FALSE and the return value is also FALSE.

If the modified data is stored successfully, the return value is TRUE (and the value of \$edit_again is a don't care).

Here is a summary of return values.

- `retval = TRUE` ==> data saved successfully
- `retval = FALSE && edit_again = TRUE` ==> re-edit the data, show the edit dialog again
- `retval = FALSE && edit_again = FALSE` ==> cancelled, do nothing

bool function `htmlpage_show_edit(&$output, $area_id, $node_id, $module, $viewonly, $edit_again, $href)` [line 115]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which this module is connected
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing is not allowed (but simply showing the content is allowed)
- *bool* **\$edit_again** if TRUE start with data from \$_POST, else use data from database
- *string* **\$href** the action property of the HTML-form, the place where data will be POST'ed

present the user with a dialog to modify the content that is connected to node \$node_id

this prepares a dialog for the user filled with existing data (if any), possibly allowing the user to modify the content. If the flag \$viewonly is TRUE, this routine should only display the content rather than let the user edit it. If the flag \$edit_again is TRUE, the routine should use the data available in the \$_POST array, otherwise it should read the data from the database (or wherever the data comes from). The parameter \$href is the place where the form should be POST'ed.

The dialog should be added to the \$output object. Useful routines are:

```
$output->add_content($content): add $content to the content area  
$output->add_message($message): add $message to the message area (feedback to the user)  
$output->add_popup_bottom($message): make $message popup in the browser after loading the page (uses
```

```
javascript)  
$output->add_popup_top($message): make $message popup in the browser before loading the page (uses  
javascript)
```

htmlpage_cron.php

/program/modules/htmlpage/htmlpage_cron.php - interface to the cron-part of the **htmlpage module**

This file defines the interface with the htmlpage-module for cron. The interface consists of this function:

```
htmlpage_cron()
```

This function is called whenever cron determines that it is time to perform this function.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_cron.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** change this stub into a real cron function.
- **License** [GNU AGPLv3+Additional Terms](#)

bool function htmlpage_cron(\$keywords, \$areas) [*line 42*]

Function Parameters:

- **\$keywords**
- **\$areas**

routine that is called periodically by cron

htmlpage_install.php

/program/modules/htmlpage/htmlpage_install.php - installer of the htmlpage module

This file contains the htmlpage module installer. The interface consists of these functions:

```
htmlpage_install(&$messages,$module_id)
htmlpage_upgrade(&$messages,$module_id)
htmlpage_uninstall(&$messages,$module_id)
htmlpage_demodata(&$messages,$module_id,$config,$manifest)
```

These functions can be called from the main installer and/or admin.php.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: htmlpage_install.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function htmlpage_demodata(&\$messages, \$module_id, \$config, \$manifest, \$configuration) [*line 91*]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$module_id** the key for this module in the modules table
- *array* **\$configuration** pertinent data for the new website + demodata foundation
- *array* **\$manifest** a copy from the manifest for this module
- **\$config**

add demonstration data to the system

this routine is a no-op because all htmlpage demodata is already created in the main demodata-routine in /program/install/demodata.php. This routine is retained here as an example alias placeholder.

bool function `htmlpage_install(&$messages, $module_id)` [*line 50*]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$module_id** the key for this module in the modules table

install the module

this routine installs the module. For this module there is nothing to install, so we simply return success. (The appropriate table is already created based on the tabledefs).

bool function `htmlpage_uninstall(&$messages, $module_id)` [*line 73*]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$module_id** the key for this module in the modules table

uninstall the module

bool function `htmlpage_upgrade(&$messages, $module_id)` [*line 61*]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$module_id** the key for this module in the modules table

upgrade the module

htmlpage_manifest.php

/program/modules/htmlpage/htmlpage_manifest.php - description of the htmlpage module

This file defines the various components of the htmlpage-module such as the names of the various scripts and version information. This file is used when this module is installed.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_manifest.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

htmlpage_search.php

/program/modules/htmlpage/htmlpage_search.php - interface to the search-part of the htmlpage module

This file defines the interface with the htmlpage-module for searching content. The interface consists of this function:

```
htmlpage_search($keywords,$areas)
```

This function is called whenever data is searched.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_search.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **TODO** change this stub into a real search function, with limits on the number of results, an offset where to start and perhaps even a time limit. for now this always returns an empty array
- **License** [GNU AGPLv3+Additional Terms](#)

bool|array function htmlpage_search(\$keywords, \$areas) [line 46]

Function Parameters:

- *string|array* **\$keywords** one or more keywords to search for
- *int|array* **\$areas** one or more \$area_id's to search

search the content of the htmlpage linked to node \$node_id

htmlpage_view.php

/program/modules/htmlpage/htmlpage_view.php - interface to the view-part of the **htmlpage module**

This file defines the interface with the htmlpage-module for viewing content. The interface consists of this function:

```
htmlpage_view(&$output,$area_id,$node_id,$module)
```

This function is called from /index.php when the node to display is connected to this module.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_view.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function htmlpage_view(&\$theme, \$area_id, \$node_id, \$module) [*line 46*]

Function Parameters:

- *object* **&\$theme** collects the (html) output
- *int* **\$area_id** identifies the area where \$node_id lives
- *int* **\$node_id** the node to which this module is connected
- *array* **\$module** the module record straight from the database

display the content of the htmlpage linked to node \$node_id

htmlpage_tabledefs.php

/program/modules/htmlpage/install/htmlpage_tabledefs.php - data definition for module

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage_tabledefs.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

htmlpage.php

/program/modules/htmlpage/languages/en/htmlpage.php - translated messages for module (English)

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

htmlpage.php

/program/modules/htmlpage/languages/nl/htmlpage.php - translated messages for module (Dutch)

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package wasmod_htmlpage Classes

Class ModuleHtmlpage

[line 30]

/program/modules/htmlpage/htmlpage.class.php - module for plain HTML-pages

This file defines a class for dealing with plain HTML-pages. It is derived from the base class Modules.

- **Package** wasmod_htmlpage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: htmlpage.class.php,v 1.1.1.1 2011-02-01 13:01:35 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

string function ModuleHtmlpage::get_content(\$node_id) *[line 37]*

Function Parameters:

- *int* **\$node_id** identifies the node

get the actual content for node \$node_id

Package wasmod_mypage Procedural Elements

mypage.php

/program/modules/mypage/languages/en/mypage.php - translated messages for module (English)

- **Package** wasmod_mypage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: mypage.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

mypage.php

/program/modules/mypage/languages/nl/mypage.php - translated messages for module (Dutch)

- **Package** wasmod_mypage
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: mypage.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

mypage_admin.php

/program/modules/mypage/mypage_admin.php - management interface for module

This file defines the administrative interface to this module. The interface consists of the following four functions.

```
mypage_disconnect(&$output,$area_id,$node_id,$module)
mypage_connect(&$output,$area_id,$node_id,$module)
mypage_show_edit(&$output,$area_id,$node_id,$module,$viewonly,$edit_again,$href)
mypage_save(&$output,$area_id,$node_id,$module,$viewonly,&$edit_again)
```

These functions are called from pagemanagerlib.php whenever necessary.

- **Package** wasmod_mypage
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: mypage_admin.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function mypage_connect(&\$output, \$area_id, \$node_id, \$module) [*line 72*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which we need to connect
- *array* **\$module** the module record straight from the database

connect this module to a node

this makes the link between the node \$node_id in area \$area_id and this module. It depends on the module what this function should do. Often it simply boils down to a no-op returning TRUE to indicate success.

bool function mypage_disconnect(&\$output, \$area_id, \$node_id, \$module) [*line 53*]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node from which we need to disconnect
- *array* **\$module** the module record straight from the database

disconnect this module from a node

this breaks the link between the node \$node_id in area \$area_id and this module. It depends on the module what this function should do. Often it simply boils down to a no-op returning TRUE to indicate success.

bool function mypage_save(&\$output, \$area_id, \$node_id, \$module, \$viewonly, &\$edit_again, \$edit_again) [line 154]

Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which we need to connect
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing and hence saving is not allowed
- *bool* **\$edit_again** set to TRUE if we need to edit the content again, FALSE otherwise
- **&\$edit_again**

save the modified content data of this module linked to node \$node_id

this validates and saves the data that was submitted by the user. If validation fails, or storing the data doesn't work, the flag \$edit_again is set to TRUE and the return value is FALSE.

If the user has cancelled the operation, the flag \$edit_again is set to FALSE and the return value is also FALSE.

If the modified data is stored successfully, the return value is TRUE (and the value of \$edit_again is a don't care).

Here is a summary of return values.

- `retval = TRUE` ==> data saved successfully
- `retval = FALSE && edit_again = TRUE` ==> re-edit the data, show the edit dialog again
- `retval = FALSE && edit_again = FALSE` ==> cancelled, do nothing

bool function mypage_show_edit(&\$output, \$area_id, \$node_id, \$module, \$viewonly, \$edit_again, \$href) [line 105]
Function Parameters:

- *object* **&\$output** collects the html output (if any)
- *int* **\$area_id** the area in which \$node_id resides
- *int* **\$node_id** the node to which this module is connected
- *array* **\$module** the module record straight from the database
- *bool* **\$viewonly** if TRUE, editing is not allowed (but simply showing the content is allowed)
- *bool* **\$edit_again** if TRUE start with data from \$_POST, else use data from database
- *string* **\$href** the action property of the HTML-form, the place where data will be POST'ed

present the user with a dialog to modify the content that is connected to node \$node_id

this prepares a dialog for the user filled with existing data (if any), possibly allowing the user to modify the content. If the flag \$viewonly is TRUE, this routine should only display the content rather than let the user edit it. If the flag \$edit_again is TRUE, the routine should use the data available in the \$_POST array, otherwise it should read the data from the database (or wherever the data comes from). The parameter \$href is the place where the form should be POST'ed.

The dialog should be added to the \$output object. Useful routines are:

```
$output->add_content($content): add $content to the content area
$output->add_message($message): add $message to the message area (feedback to the user)
$output->add_popup_bottom($message): make $message popup in the browser after loading the page (uses javascript)
$output->add_popup_top($message): make $message popup in the browser before loading the page (uses javascript)
```


Package wastheme_frugal Procedural Elements

frugal_install.php

/program/themes/frugal/frugal_install.php -- installer of the frugal theme

This file contains the frugal theme installer. The interface consists of these functions:

```
frugal_install(&$messages,$theme_id)
frugal_upgrade(&$messages,$theme_id)
frugal_uninstall(&$messages,$theme_id)
frugal_demodata(&$messages,$theme_id,$config,$manifest)
```

These functions can be called from the main installer and/or admin.php.

- **Package** wastheme_frugal
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: frugal_install.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

bool function frugal_demodata(&\$messages, \$theme_id, \$config, \$manifest, \$configuration) [line 230]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$theme_id** the key for this theme in the themes table
- *array* **\$configuration** pertinent data for the new website + demodata foundation
- *array* **\$manifest** a copy from the manifest for this theme

- **\$config**

add demonstration data to the system

this routine is a no-op because all frugal demodata is already created in the main demodata-routine in /program/install/demodata.php. This routine is retained here as an example alias placeholder.

bool function frugal_install(&\$messages, \$theme_id) [line 45]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$theme_id** the key for this theme in the themes table

install the theme

bool function frugal_uninstall(&\$messages, \$theme_id) [line 212]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$theme_id** the key for this theme in the themes table

uninstall the theme

bool function frugal_upgrade(&\$messages, \$theme_id) [line 200]

Function Parameters:

- *array* **&\$messages** collects the (error) messages
- *int* **\$theme_id** the key for this theme in the themes table

upgrade the theme

frugal_manifest.php

/program/themes/frugal/frugal_manifest.php - description of the frugal theme

This file defines the frugal theme. This file is used when this theme is installed.

- **Package** wastheme_frugal
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: frugal_manifest.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

frugal.php

/program/themes/frugal/languages/en/frugal.php - translated messages for theme (English)

- **Package** wastheme_frugal
- **Author** Peter Fokker < peter@berestijn.nl>
- **Version** \$Id: frugal.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

frugal.php

/program/themes/frugal/languages/nl/frugal.php - translated messages for theme (Nederlands)

- **Package** wastheme_frugal
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: frugal.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package wastheme_frugal Classes

Class ThemeFrugal

[line 27]

/program/themes/frugal/frugal.class.php - the class that comprises the most minimal theme

- **Package** wastheme_frugal
- **Author** Peter Fokker < peter@berestijn.nl >
- **Version** \$Id: frugal.class.php,v 1.1.1.1 2011-02-01 13:01:36 pfokker Exp \$
- **Copyright** Copyright (C) 2008-2011 Ingenieursbureau PSD/Peter Fokker
- **License** [GNU AGPLv3+Additional Terms](#)

Package default Procedural Elements

spellchecker.php

- **Package** default

void function error_handler(\$err) [*line 63*]

Function Parameters:

- **\$err**

void function escape_quote(\$str) [*line 57*]

Function Parameters:

- **\$str**

void function print_checker_results() [*line 69*]

void function print_suggs_elem(\$suggs, \$index, \$text_input_idx) [*line 43*]

Function Parameters:

- **\$suggs**
- **\$index**
- **\$text_input_idx**

void function print_textindex_decl(\$text_input_idx) [*line 31*]

Function Parameters:

- **\$text_input_idx**

void function print_textinputs_var() [*line 22*]

void function print_words_elem(\$word, \$index, \$text_input_idx) [*line 37*]

Function Parameters:

- **\$word**
- **\$index**
- **\$text_input_idx**

fckeditor.php

- **Package** default

include_once ["fckeditor_php4.php"](#)*[line 29]*

include_once ["fckeditor_php5.php"](#)*[line 31]*

fckeditor_php4.php

- **Package** default

boolean function FCKeditor_IsCompatibleBrowser() [*line 34*]

Check if browser is compatible with FCKeditor. Return true if is compatible.

Package default Classes

Class FCKEditor *[line 74]*

- **Package** default

FCKEditor::\$BasePath

string = [line 88]

Path to FCKEditor relative to the document root.

FCKEditor::\$Config

array = [line 122]

This is where additional configuration can be passed.

Example: `$oFCKEditor->Config['EnterMode'] = 'br';`

FCKEditor::\$Height

mixed = [line 102]

Height of the FCKEditor. Examples: 400, 50%

FCKEditor::\$InstanceName

string = [line 82]

Name of the FCKEditor instance.

- **Access** protected

FCKEditor::\$ToolbarSet

string = [line 108]

Name of the toolbar to load.

FCKEditor::\$Value

string = [line 114]

Initial value.

FCKEditor::\$Width

mixed = [line 95]

Width of the FCKEditor. Examples: 100%, 600

Constructor *void* function FCKEditor::FCKEditor(\$instanceName) [line 130]

Function Parameters:

- *string* **\$instanceName**

Main Constructor. Refer to the `_samples/php` directory for examples.

void function FCKEditor::Create() [line 146]

Display FCKEditor.

string function FCKEditor::CreateHtml() [line 156]

Return the HTML code required to run FCKEditor.

string function FCKEditor::EncodeConfig(\$valueToEncode) [line 253]

Function Parameters:

- *string* **\$valueToEncode**

Encode characters that may break the configuration string generated by `GetConfigFieldString()`.

- **Access** protected

string function FCKEditor::GetConfigFieldString() [*line 222*]

Get settings from Config array as a single string.

- **Access** protected

boolean function FCKEditor::IsCompatible() [*line 211*]

Returns true if browser is compatible with FCKEditor.

Appendices

Appendix A - Class Trees

Package wascore

AclManager

- [AclManager](#)

AdminOutput

- [AdminOutput](#)

Area

- [Area](#)

AreaManager

- [AreaManager](#)

ConfigAssistant

- [ConfigAssistant](#)

DatabaseMysql

- [DatabaseMysql](#)

DatabaseMysqlResult

- [DatabaseMysqlResult](#)

Email

- [Email](#)

FileManager

- [FileManager](#)

GroupManager

- [GroupManager](#)

Language

- [Language](#)

Module

- [Module](#)
 - [ModuleHtmlpage](#)

Node

- [Node](#)

PageManager

- [PageManager](#)

Theme

- [Theme](#)
 - [ThemeFrugal](#)

TranslateTool

- [TranslateTool](#)

Useraccount

- [Useraccount](#)

UserManager

- [UserManager](#)

Zip

- [Zip](#)

Package default

FCKeditor

- [FCKeditor](#)

FCKeditor

- [FCKeditor](#)

Package waslang_en

Package waslang_nl

Package wasinstall

InstallWizard

- [InstallWizard](#)

Package wasmod_guestbook

Package wasmod_htmlpage

ModuleHtmlpage

- [Module](#) (Different package)
 - [ModuleHtmlpage](#)

Package wasmod_mypage

Package wastheme_frugal

ThemeFrugal

- [Theme](#) (Different package)

- [ThemeFrugal](#)

Appendix B - README/CHANGELOG/INSTALL

FAQ

/program/FAQ.txt

\$Id: FAQ.txt,v 1.1.1.1 2011-02-01 13:00:05 pfokker Exp \$

This file will contain answers to frequently asked questions.

Q1: Can I temporarily close my website?

A1: Yes. If a file called 'maintenance.html' exists in the top level directory (i.e. the directory holding the main entry points and also the file 'config.php'), a website visitor will be redirected to that file 'maintenance.html' instead of the real site.

Q2: What is this message "condition code 010"?

A2: It means that the site is not yet configured. Point your browser to /program/install.php to start the installation wizard and follow the instructions.

Q3: What is "condition code 050"?

A3: This indicates that the version of the program (the .PHP-files) is not the same as the structure of the database. Usually this means that an update needs to be done in the database, changing the database structure.

TODO

/program/TODO.txt

\$Id: TODO.txt,v 1.1.1.1 2011-02-01 13:00:05 pfokker Exp \$

2008-01-29

- convert /program/*.txt to DOS-text
- decide on exact license conditions

2008-02-01

- add an explanation to the manual about \$CFG->datadir within the document root and a 'difficult' name that cannot be guessed.

2008-02-07

OK should we document \$CFG->debug? this is a feature that is handy during development, but it should be 'off' on a production server
Answer: this is now documented.

- how can we discriminate between a call from the command line and a call via a web interface? Is it a mere check of \$_SERVER['REMOTE_ADDR']? (You _never_ have a remote address when working from the command line, do you?). Would be handy in case of cron.php.

2008-02-14

-- make a separate subtree for testing purposes. I don't think that that test environment should be published for end users (but it should be for developers). Maybe a subdirectory /devel/test/ and a corresponding /devel/test/test.php?

2008-02-19

-- can it be possible to have a page visible in navigation and not available to serve? I'm inclined to say yes: if a node only has a href, no actual data is available in the database. However, should we treat this as a call nevertheless and count the page view just before we redirect via header()? Mmm....

2008-03-22

OK Protect password fields with AUTOCOMPLETE="off" (see http://www.owasp.org/index.php/Guide_to_Authentication)

OK Count the number of failed login-attempts within a certain period of time

OK On-time password reset should time out in a short time, e.g. 15 minutes (ok, make it 30 minutes)

OK Should there be password requirements (min 1 digit, min 1 capital, minimum length?): YES one of each.

OK Add delay in user validation (3 seconds per attempt)

OK Should login-forms have short-lived unique identifiers? (prevents brute force attacks?) Methinks that our blacklist facility (8 minutes after more than 10 failed attempts) is enough.

2008-04-25

-- Add to cronjobs: cleanup of expired sessions + logging of the fact

OK Should we add the number of calls + length of session in seconds to the logout message? What then if the session expires without a proper logoff? A: added this for regular logouts

-- Add to cron: removal of obsolete login_failures? It should be auto-cleaning but if a user fails once and never returns, the failure will be there forever.

OK Get rid of the 'trigger_error()' calls now that we have logger(), maybe except errors in the database routines..

-- Logrotate for log_messages table? How? Send a summary to the webmaster before deletion?

2008-04-27

-- we should be able to optimise with relative links (sometimes) if the hostname part of \$CFG->www and \$CFG->progwww are the same. Example: when pointing to the source of an image the src would be {\$CFG->progwww}/graphics/foo.png, i.e. including the 'http://www.exemplum.websiteatschool.org' part. If \$CFG->www also starts with that string, we can leave it out in the src property: src="/program/graphics/foo.png".

Here is a generic example from RFC3986:

```
foo://example.com:8042/over/there?name=ferret#nose
 \_/   \_|   \_|   \_|   \_|   \_|   \_|
 |       |       |       |       |
scheme authority path query fragment
```

If \$CFG->www and \$CFG->progwww have the same scheme and authority, links could be made relative, e.g.
 and

 instead of prefixing with an explicit "foo://example.com:8042/". What if the scheme and authority are NOT the same?

2008-05-01

-- About the installer: if the installer is not able to write 'config.php' to the target directory, it should be possible to 'download' a .ZIP-file with this config.php, and perhaps also with index.php, cron.php, admin.php? Unzip that in the appropriate place and you are in business.

2008-08-01

-- How about storing ETags and be friendly to caches/proxies?
-- How about using index.php/aaa/nnn/friendly-title-of-page-for-bookmark instead of index.php?area=aaa&node=nnn
That too is cache/proxy-friendly...

2008-09-29

OK How about adding an extra parameter to manual.php in order to 'deep link' to the manual? Example: if the user has selected the pagemanager in admin.php, should the help button be calling 'manual.php?language=en&job=pagemanager' or something? Done: additional parameter 'topic' is added, at least from page manager. Could be employed from start centre too if necessary.

2008-10-01

-- Should there be an attribute 'is_root' or 'is_guru' in the users table? E.g. if a node is owned by some user, only that user is allowed to remove the read-only bit. Shouldn't there be a way to override that other than by using phpMyAdmin to manipulate the database directly?

2008-10-10

-- Edit permissions not only depend on the user's permission bits but also on the read-only attribute of a node.
-- We should isolate the check-permission routines because they are used when creating icons and also when executing the corresponding function.

2008-10-20

-- get some decent graphics for button_save and button_cancel and other buttons...

2008-10-28

OK How about changing all where clauses with "(value = 'NULL')\" into \"(value IS NULL)\" to be more SQL-standard compliant? Done, in databaselib.php and function db_where_clause().
-- So how about recovering from a crashed browser with an inaccessible session? It is very annoying to be locked out in the cold if the browser crashed and I immediately login again using the same credentials and still not be able to carry on...
Should we add the time of the lock, too? That would give the other user an educated guess about the odds that the other session is still active. Mmmmm...

2008-11-12

-- the output object AdminOutput should have something like a 'funnel mode'. This mode should disable all navigation links and other

links that could distract and/or seduce the user to leave the page. Main use: prevent locked records hanging around without the user actively editing that record.

2008-11-20

- the Theme should have a property 'date/time last modified' so a module can update it via `module_view()`.
- how to create an alert based on edit content (or not)? How can we tell that the content actually changed? Is there a 'dirty' bit somewhere? Should we return a status insted of simply true/false?

2008-12-10

- How about the visibility of nodes when in preview mode????

2009-02-25

- do we actually `_USE_` the field `muser_id` in the page manager? we do in the area manager.

2009-02-26

- How about adding transactions (rollback, commit)
See function that deletes areas.
Q: How well is MySQL suited for transactions?
A: Mmmmm.... Which version? Which storage engine?

2009-05-31

- Why not use the 'name' field in the `dialogdef[]` arrays?
`$dialogdef = array('fullname' => array('name' => 'fullname', 'type' => F_STRING, ...));` Much more convenient when saving data.
- Doublecheck for `htmlspecialchars()` when displaying information from the database in a dialog: a username like say '`<u>ser</u>name`' should be converted to '`<u>ser</u>name`' before being sent to the browser.

2009-06-04

- we should re-analyse the repercussions of the option to delete user accounts. If only the `user_id` is logged, it is difficult to figure out whodunnit if the account has already disappeared, not to mention breaking the constraint of the foreign key. This is also a problem for tables with `cuser_id` and `muser_id`...

2009-09-22

- We should have a graphics designer restyle all icons etc. I'm no good at that.

2009-12-04

- Should we make the permissions 0700 configurable, eg in `areamanager`?

2010-09-27

- There is a potential security issue with relative paths: the check on `'../'` is inconclusive if the `$path` is encoded in UTF-8: the overlong sequence `2F C0 AE 2E 2F` eventually yields `2F 2E 2E 2F` or `'../'`. Reference: RFC3629 section 10. Applies to `main_file.php` and also `filemanager.class.php`.

2010-12-08

-- Should we add a configurable favicon to themes as a config option
(defaulting to /program/graphics/favicon.ico)?

LICENSE

/program/LICENSE.txt

\$Id: LICENSE.txt,v 1.1.1.1 2011-02-01 13:00:08 pfokker Exp \$

The 'LICENSE AGREEMENT for Website@School'
can be found in the file 'license.html'
in the program directory or on-line at
<http://websiteatschool.org/license.html>

The necessary 'Appropriate Legal Notices'
can be found in the file 'about.html'
in the program directory.

[eof]

README

/program/README.txt

\$Id: README.txt,v 1.1.1.1 2011-02-01 13:00:08 pfokker Exp \$

This is the README for the Website@School Content Management System.

See INSTALL.txt for installation instructions.
See LICENSE.txt for license information.
See HISTORY.txt for the project history.
See CREDITS.txt for information about contributors to Website@School.
See FAQ.txt for answers to frequently asked questions.
See CHANGES.txt for significant changes between revisions.
See TODO.txt for a list of things that still need to be done.

The homepage of the Website@School Project is <http://websiteatschool.org>.

HISTORY

/program/HISTORY.txt

\$Id: HISTORY.txt,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$

History of Website@School

=====

Website@School started in Amsterdam in 2002 and as they say: the rest is history.

INSTALL

/program/INSTALL.txt
\$Id: INSTALL.txt,v 1.1.1.1 2011-02-01 13:00:10 pfokker Exp \$

Quick Install of Website@School

=====

1. Download the latest distribution version of Website@School (either as a .ZIP-file or a tar.gz-file.).
2. Unpack the distribution file somewhere on your webserver. This will yield a number of files (index.php, admin.php, etc.) in the CURRENT directory. Also, a subdirectory 'program' with all the other program files will be created.
3. Start the installation process by pointing your browser to 'program/install.php' and follow the directions.
4. Your new website is ready.

CHANGES

/program/CHANGES.txt
\$Id: CHANGES.txt,v 1.2 2011-02-01 14:34:34 pfokker Exp \$

2011-02-01

Today the complete development of WebsiteAtSchool was transferred to CVS at the development site BerliOS (<http://developer.berlios.de>), using the current version (which we used to call 0.1.7).

We now start with a new phase in the development, i.e. a publicly visible CVS-server and official releases etc. This is a good opportunity to bump both the internal version and the external version (release) and start with a more or less clean slate.

Furthermore, we will be generating a first 'official' release today.

Changes in /program/version.php:

- Bumped WAS_RELEASE from 0.1.7 to 0.90.0
- Bumped WAS_VERSION from 2010122100 to 2011020100
- Bumped WAS_RELEASE_DATE from 2010-01-20 to 2011-02-01

Changes in /program/lib/updatelib.php:

- Added logic to update existing database versions to 2011020100

```

*****
*****
*****
***** BELOW THIS LINE ONLY OLD AND OBSOLETE CHANGES *****
*****
*****
*****
*****

```

2011-02-01

- changed development colours to something more appropriate

2011-01-20

- completely overhauled version of devel/tools/makedist.sh:
we can now generate 'official' releases too
- updated copyright/license message in scripts in devel/tools
- bumped release from 0.1.6 to 0.1.7 just to test the release procedure. After checking in version.php we need to add a revision tag with 'cvs rtag release-0_1_7 was' and see what happens.

2011-01-12

- added a simple counter/index to translate tool: makes it easier to refer to a particular translate string on the phone
- added a devel tool to identify changed and added strings between versions in a convenient single HTML-page (langdiff.sh in the tools directory)
- corrected a stupid typo in English and Dutch translations
- bumped version from 0.1.5 to 0.1.6

2011-01-10

- Changed bottom line in main_admin.php: we now only display the execution time, # of queries and TOD when in debug-mode.
- Removed the (empty demo-like) stubs in the group manager and user manager indicating configurable modules per user/group. This makes it easier to create realistic screenshots for the upcoming manual.
- Bumped release from 0.1.4 to 0.1.5

2010-12-20

- completed the automatic execution of the update manager (called whenever there is a mismatch in the core version)
- added update manager to tools menu
- bugfix: removed double escape in version check icon in start center
- bumped release from 0.1.3 to to 0,1,4
- bumped version from 2010120800 to 20101221 (just for testing purposes)

2010-12-17

- started with support for automatic update wizard (driven by internal version number mismatch)
- bumped version to 0.1.3

2010-12-13

- bugfix: missing closing '>' in buttons in install.php
- bumped release to 0.1.2
- bugfix: error with deleting of sections/nodes in pagemanager.

2010-12-10

- bugfix: if CFG->www_short is empty, the link to the public

- site was broken [main_admin.php]
- bumped release to 0.1.1

2010-12-08

- mass update of all source files, now with the correct copyright text and link to license.html etc. (continued)
- renamed DatabaseResult to DatabaseMysqlResult to prevent name clashes once DatabasePostgresql and DatabasPostgresqlResult once these are written
- cleaned up various files (removed obsolete chunks of dead code)
- added the 'registered trademark' message to the loginlib
- removed the old (wrong) logo from the standard theme
- updated base.css to increase height of quickbottom div
- added a rel link to /program/graphics/favicon.ico (in main_admin.php)
- bumped external version from 0.0.7 to 0.1.0
- bumped internal version from 2010092700 to 2010120800

2010-12-07

- cleanup of init.php; streamline the error_exit() routine
- mass update of all source files, now with the correct copyright text and link to license.html etc.
- various cosmetics
- added link to on-line license.html in LICENSE.txt too

2010-12-02

- added the final version of license.html with all the legalese
- added /program/about.html with 'Appropriate Legal Notices'
- modified the installer to check for the exact version of license.html
- added global definition: WAS_ORIGINAL in version.php
- added some more logic to the online version check
- bumped release from 0.0.6 to 0.0.7
- added a few graphics files for 'powered by' and 'based on'
- added a new logo file of 284x71 (instead of 248x53)
- incorporated the 'powered by' and 'based on' graphics in install.php
- incorporated new logo in install.php
- header height in base_admin.css from 53 to 71 pixels for new logo
- added link to about.html via function appropriate_legal_notices() in install.php
- added link to about.html via function appropriate_legal_notices() in main_admin.php
- modified the translatable text in footer: we now force the English 'powered by' (or 'based on') even in other languages.
- added an explanation for creating modified versions in '/program/about.html'
- also changed the address for new users from schoutdi@knoware.nl to online@websiteatschool.org

2010-10-28

- raised the priority of logmessage when a virus is detected in filemanager.virusscan()

2010-10-27

- we now use UTF-8 in the default Theme class; send it to browser via http-headers
- stupid typo in Theme class corrected (bugfix)
- new function: log view (in tools menu)
- added two more job permissions and changed existing: we now have separate permissions for translate tool,

backup an logview.

2010-10-26

- minor cosmetics: we now have a database backup named host-database-prefix-date-time (either .zip or .sql)
- we now have a 'hidden' feature that allows for downloading the uncompressed .sql by specifying 'download=sql' instead of 'download=zip'.

2010-10-21

- bugfix in rfc2047_qstring(): we now keep UTF8-tails together with the first octet in the sequence in the same 'encoded-word'
- completed backup function in tools menu

2010-10-20

- implemented a (database-specific) database/SQL-dump in mysql class
- first draft of database dump downloader in tools (no check on user perms yet)
- minor cosmetics

2010-10-19

- moved a few mail-related routines from waslib to email.class
- adapted loginlib: we now use email class to send mail
- added a header implying that we send UTF-8 in login dialog too
- added an extra download feature: file.php/websiteatschool/languages now returns a ZIP-file with all user-defined translations of active languages

2010-10-18

- added a supersimple mailer class in email.class.php
- adapted translatetool to use this new email class
- minor cosmetics

2010-10-17

- fixed some bugs in filemanager.class.php
- preliminary changes in interfacing to mail() command (viruscan)
- cosmetics

2010-10-15

- implementation of RFC2047-style quoted-printable mail header extensions in translate tool

2010-10-11

- implementation of conversion to quoted printable according to RFC2045 section 6.7 (waslib.php)
- adapted routine to submit translations to deal with quoted printable message
- (finally) converted the program's native language to UTF-8, at least in AdminOutput in main_admin.php (we still need to deal with UTF-8 in the database)

2010-10-06

- translatetool.class.php: we can now really save user translation files under CFG->datadir/languages/ (submit of translations is still work in progress)
- (later) initial version of sending translation works (more or less).
- added a cache reset to \$LANGUAGE to force re-read of the userfile just written

2010-10-05

- minor changes in language.class.php:
 - . we now cache a list of `_all_` (active+inactive) languages
 - . removed obsolete and unused routine `get_languages()`
 - . renamed `get_language_names()` to `get_active_language_names()`
- minor changes in files which relied on the old language.class.php
- nasty bug: a '.' in a fieldname yields a '_' in `$_POST`. Huh?
workaround: use a ':'

2010-10-04

- translatetool:
 - . we now have visible codes in the translation dialog.
 - . we also have fields for metadata (per full_domain)
 - . still work in progress

2010-09-30

- translatetool: we can now show a domain menu in the edit translation dialog but otherwise it is still work in progress

2010-09-29

- translatetool: we can now add new languages and edit existing ones but it still work in progress

2010-09-28

- translatetool is work in progress

2010-09-27

- changed the way languages are handled: we now store the name of the language expressed in the language itself in the languages table which required a change in the tabledefs
- bumped version from 2010070700 to 2010092700 because of the changes in the database (table definition of 'languages').

2010-09-23

(after the summer break)

- bugfix (finally) in language class where valid translations were overwritten in memory when another phrase key was not found.

2010-07-08

- we now have a distinction between browsing files limited by filename extension and uploading files limited by extension.
bottom line: in the file/image/flash browser the user can only see and upload selected files, in the file `_manager_` the user can upload selected files but see `_all_` files, including 'forbidden' files (eg. some random script uploaded by an intruder)
- new feature: if a file is detected (in filemanager) which would currently not pass the test for allowable upload, the entry in the filemanager listing is displayed with the CSS error class and the preview links are completely disabled, preventing the user from accidentally previewing a rogue file.

2010-07-07

- now suppress menu too in filebrowser/imagebrowser (via .CSS)
- started with support for new job 'flashbrowser' (cousin of 'imagebrowser')
- added support for three new configuration parameters that limit the allowable file extensions (as a shorthand for allowable file types)
- bumped version from 2010063000 to 2010070700 because of the three new configuration options in the installer

2010-07-02

- initial version of thumbnail-based file/image browser (still incomplete)

2010-07-01

- added check on GD in installer
- added a hint to print the summary page in installer

2010-06-30

- bumped WAS_VERSION from 2010052400 to 2010063000 because of a typo last week (it should have read 2010062400 but the actual value was 2010052400). Also, the new default value for 'thumbnail_dimension' decreased from 150 to 100 pixels.
- added support for thumbnails in the basic style sheet (based on the new default dimension of 100).

2010-06-24

- added generation of thumbnails to file upload routine in file manager
- added more about limits (file size, upload size) in file upload explanation
- minor cosmetics
- bumped WAS_VERSION from 2010051300 to 2010062400 because of new configuration option 'thumbnail_dimension'

2010-06-22

- re-arranged code and added new file /program/lib/filelib.php

2010-06-15

- filemanager extended so it can double as a filebrowser via job=filebrowser and image browser via job=imagebrowser
- linked FCKEditor to the filebrowser and imagebrowser

2010-06-14

- draft version of link browser combined in filemanager (filebrowser)

2010-05-04

- done with file upload except for 1 small thing (sanitise file type)
- minor cosmetics here and there

2010-05-13

- renamed new parameter 'files_upload_count' to 'upload_max_files'
- bumped WAS_VERSION from 2010051200 to 2010051300 because of this
- minor cosmetics here and there
- implemented F_FILE in dialoglib

2010-05-12

- added compatibility check for Safe Mode (should be 'Off') in install.php
- added support for clamscan anti-virus in install.php
- added extra configuration options in site config for clamscan
- added configuration option for maximum number of simultaneous uploads
- bumped WAS_VERSION from 2010010700 to 20100512 because of extra site configuration options

2010-04-14

- almost done with filemanager; what remains is file upload implementation

2010-04-13

- added file delete confirmation dialog to filemanager

2010-03-31

- filemanager: added a Javascript one-liner to select all files
- filemanager: we can now add (create) subdirectories

2010-02-13

- filemanager:
 - . directory listing layout done
 - . sort option for three columns (ascending or descending)
 - . human-readable filesize
 - . various stubs for delete and add file/dir
- still work in progress though

2010-02-10

- filemanager: navigation basically works but still: work in progress

2010-02-08

- filemanager is still work in progress

2010-02-05

- added some more navigation to file manager

2010-02-03

- traded filemanagerlib.php for filemanager.class.php

2010-01-28

- for now done with main_file.php: we can serve files from the data directory using the correct (?) mime types etc.

2010-01-20

- Oops. Bug in tabledefs: there was no unique index on areas.path. Fixed.
- Done with download of source code: we now recognise file.php/websiteatschool/program (program code) and file.php/websiteatschool/manual (manual in current language)
- More oops. Another bug in tabledefs: we didn't have the unique index on path for users and groups either. Fixed.
- Busy with file.php: validation and access control done.

2010-01-19

- started on file.php and download of source in .ZIP-file

2010-01-13

- created a new class Zip (in /program/lib/zip.class.php) which allows for creating ZIP-archives on the fly, including Deflate compression, file and archive type comments and zipping from memory or from existing file.

2010-01-09

- removed a lot of superfluous files from /program/lib/fckeditor
- suppressed the option 'fcksource=true' in dialoglib: the user that sets that parameter in \$_GET gets the plain text area instead of the source-variation of FCKEditor (which is mostly deleted from /program/lib/fckeditor and wouldn't work anyway).

2010-01-08

- added the full tarball FCKEditor_2.6.5.tar.gz to devel/imports so we have the original source nearby
- added type F_RICHTEXT as a gateway to the FCKEditor in dialoglib

- first steps to actually incorporate FCKeditor 2.6.5 in directory /program/lib/fckeditor ia CVS import function. We consider the extensions gif png swf pfx and fla to be binary files, the rest done via -ko.

2010-01-07

- added view-only field in edit user that shows the data directory name
- removed the 'advanced' option from the edit user menu because there is no 'advanced' dialog (yet)
- added editor selection to basic user properties, choices are: plain and fckeditor
- added the logic for groups path parameter
- added r/o view of data dir name in edit function in areamanager
- added r/o view of data dir name in edit function in usermanager
- added r/o view of data dir name in edit function in groupmanager
- corrected some typos
- added unique index on path field in areas table
- bumped version from 2010010600 to 2010010700 because of changed tabledefinition
- added the data path to the add area dialog
- update demodata so we have readable area data dirs now instead of numbers

2010-01-06

- we now also make subdirectories 'users' and 'groups' in the data directory (next to 'areas' and 'languages') in the installer
- we no longer record the words 'areas' and 'users' and 'groups' in the path fields in the corresponding tables, dirs are strictly separated now.
- added global configuration option for generating friendly URLs instead of querystrings
- added routine to sanitise filenames to waslib (used in creating datadirectories for areas, users and groups)
- we now check to see if the user tries to install with a username that is also used in the demodata
- we now actually create the datastructures in the datadir for all demo-data users, groups and also the areas
- bumped version from 2009120300 to 2010010600 because of changed tabledefinitions
- added more translation texts for new config items 'friendly_url' and 'editor'
- fixed a few typos

2009-12-22

- done with pagemanager class
- removed obsoleted pagemanagerlib.php

2009-12-21

- almost done with pagemanager class

2009-12-19

- 'playing' with stylesheet: we now see line-through on 'dimmed' links in high-visibility mode (works for funnel mode)

2009-12-18

- added funnel mode support to AdminOut

2009-12-17

- very busy re-arranging code in page manager

2009-12-08

- changed the `has_...permissions()` routines in `useraccount`:
we now test for 1 or more permissions, not an exact match
anymore: `($perm & $mask) != 0` versus `($perm & $mask) == $mask`.

2009-12-04

- finally decided on the issue of 'per-area' datadirectories:
we simply use `$CFG->datadir/areas/<area_id>`
- reworked the permissions in the area manager
- we no longer allow changes in the path property of areas;
the field isn't even shown anymore.

2009-12-03

- finally got rid of the tables `users_areas` and `users_nodes`
because they are now replaced by the `acls`-tables.
- removed fields `job_permissions` and `permissions` from `users` table
because they are now replaced by the `acls`-tables.
- implementing `acls` in the `useraccount` class (work in progress)
- bumped version from 2009102100 to 2009120300 because of
changed/removed tables
- removed `PERMISSION_VIEW` from `useraccount.class.php` because
it is now replaced by `ACL_ROLE_INTRANET_ACCESS`
- removed `PERMISSION_ADMIN` (which was added 2008-10-01) from
area manager. It still needs to go from `pagemanager`.

2009-11-30

- added an item to the CMS dialog in the installer
to query the user for a generic demonstration data password
(will be assigned to all demo accounts)

2009-11-04

- done with `demodata`: we now have a public area with
16 pages and 6 sections and a private area with 8 pages
and 3 sections, alltogether in 2 languages (en and nl).
Pfew!

2009-11-03

- `demodata()` now adds 3 areas, 4 groups and 8 users in 2 languages
- `demodata()` now also adds many nodes in 2 languages (work in progress)
- reduced the `htmlpage_demodata()` and `frugal_demodata()`
to no-ops because all `demodata` is already inserted in the main
`demodata()` routine.

2009-11-02

- finished logic for `datadirectory` name based on `dataroot`
and quasi-random subdirectory (to obfuscate the `datadir`
if it happens to be located within the document root)
- we now create an empty `index.html` in both `dataroot` and
`datadir` to prevent leakin information if `dataroot` is
located within document root.
- added creation of subtree 'languages' to `datadirectory`,
also with empty `index.html` files

2009-10-23

- re-arranged the interface via manifest-files: we now
fill the array `manifests[]` (plural), just like `tabledefs[]`

- main logic for installing is done, only thing left to do is insert demodata
- quick fix in set defaults for cms_dir

2009-10-21

- yet another changed tabledef: bumped version from 2009102000 to 2009102100
- we now install themes and languages too (not just modules)

2009-10-20

- bumped version to from 2009101600 to 2009102000 because of changed tabledefs 'modules', 'themes' and 'languages'
- added manifest files for languages 'en' and 'nl'

2009-10-19

- changed manual.php: we made a start with linking topics and subtopics to html-files under /program/{language}/
- changed handling of help_topic in install wizard (subtopics)
- we now actually create the data directory if it doesn't exist already
- we now return a link to check for new versions rather than the simple assumption 'OK' in the compatibility dialog in the install wizard

2009-10-16

- removed fields manifest_script and install_script from modules table; if necessary we can construct these from the manifest or use 'well known' script names (like 'tabledefs.php' and {\$module}_manifest).
- Bumped version from 2009061100 to 2009101600 because of change in the datadefinition of the modules table.
- We now are able to install modules via their manifest (but not yet demodata)

2009-10-15

- extra debugging in database library and mysql class
- almost done with actual installer

2009-10-13

- added more documentation to functions
- added functionality to write config.php (if filesystem permits)

2009-10-12

- added a check on already installed
- added optional version check in compatibility screen

2009-10-05

- the installer is still work in progress but all the dialogs are done

2009-10-01

- more checks/validation in the installer

2009-09-30

- re-arranged the code in install.php: we now have the InstallWizard class
- re-arranged the order of the funnel: we now require valid database credentials before we leak any information
- added delays to the database validation if something goes wrong, to scare off the spooks

2009-09-29

- added suppress_output mode to AdminOutput class

2009-09-24

- installer: work in progress

2009-09-23

- installer: work in progress

2009-09-22

- installer: work in progress
- added an ad-hoc image for an OK-button

2009-09-21

- installer: work in progress

2009-09-18

- added style for previous and next buttons to admin_base.css + icon

2009-09-10

- we now require a new password twice when adding a new user
- code cosmetics

2009-07-24

- done with pagemanager permissions in usermanager and groupmanager; we can now store and retrieve permissions on node level.
- added minor cosmetics in the expanding/collapsing of areas in aclmanager: we now always keep the section to expand/collapse visible on the current screen by setting the offset to the offset of that area
- updated translations

2009-07-23

- after a lot of time I finally got the pagemanager permissions in the ACL Manager right: we can now actually save the settings via a complex dialog (for the programmer) that deals with screens (in long lists) and also allows for opening and closing areas.
pfew

2009-07-01

- more work on the pagemanager permissions dialog, including walking the node tree in an area (work in progress)

2009-06-30

- moved pagination code to AdminOutput()

2009-06-19

- we now have fully implemented the 'related-acls' feature for the user intranet access dialog and the user admin access dialog.
- some more streamling in the install.php to navigate in tabledefs and datadump

2009-06-18

- we now also delete user associations with a group/capacity when a capacity is removed from a group in the groupmanager
- (special request) we now have a jump-menu in the theme via which a visitor can jump to any area available to her (logged-in users also see private areas if they have permission)
- added a users+groups summary to the account manager intro screen

- added support for user privileges intranet and also user privileges for admin, the latter including the related feature
- various typo's corrected

2009-06-17

- we can now manipulate a user's group memberships: view, add, delete

2009-06-16

- completed pagination routine with a limit on links to show
- changed default number of links to show to 7 (was 5)

2009-06-11

- added pagination to users list in user manager
- added configuration parameters for pagination to \$CFG via config table
- added 'memory' to users list: after editing you return to list/screen you started from (if possible)
- bump version to 2009061100

2009-06-05

- added function for sorted list of languages to \$LANGUAGE
- saving basic properties of user now works

2009-06-04

- minor tweaks in loginlib in order to reuse the password/salt-code
- added 'add user'
- added 'delete user'
- started with 'edit user'
- added more Dutch translations, again

2009-05-31

- done with acl for admin jobs
- fine-tune in group capacity menu: suppress pagemanager link if this group/capacity has no permission for page manager
- more or less done with users overview (but Dutch translations lack)

2009-05-30

- code cleanup in group manager
- more logging in group manager
- some small convenient enhancements to the install.php making it easier to keep track of the logging
- delete group function now works
- added more Dutch translations

2009-05-29

- we can now save group data, including checks on ACLs etc.

2009-05-28

- busy with implementing ACL-configuration starting with intranet permissions
- added more and more information to the demodata: now with an acls-record for every user and every group/capacity.

2009-05-26

- typo in tabledefs; bumped version to 2009052600

2009-05-22

- added a breadcrumb trail to the admin output object because we really need this to remember where we are in the account manager
- added list of users per group-capacity (task=capacityoverview)

- minor cosmetics (including correction of type in name Helen Parkhurst)
- busy with the navigation between various screens dealing with group-capacity properties

2009-05-20

- added some 55 dummy-users and 15 dummy-groups to demodata in order to 'play' with the account manager
- we now are able to add new groups, including up to 8 capacities per group

2009-05-19

- group manager: we can now show a list of existing groups and group-capacities including clickable links
- added some group data to the demodata
- bumped the version to 2009051900 to make sure that the demodata is re-created in the testversion

2009-05-18

- updated logo's, now with ®
- added new icon for account manager
- moved usermanager around together with the group manager; there is now a top-level accountmanager for users and groups

2009-05-14

- (after some two months of thinking we're back with more)
- start with better access control via 6 new tables
 - added three tables for additional group-based access control
 - obsoleted two tables (users_areas and users_nodes) and some fields in the users table: the functionality is now available via ACLs.
 - bumped versions to 0.0.5 and 2009041500
 - Deprecated the 'unsigned' attribute in tabledefs because this is a non-standard feature which MySQL happens to implement but other DBMSs might not. In general 2^{31} integers should suffice anyway.

2009-03-18

- rearranged code for calculating the URL for version checker on project's website
- added support for Bazaar Style Style Sheets on static level and on area level (not yet on node level; todo)

2009-03-17

- fixed bug in dialog_validate(): viewonly fields should not need to be validated because they are not supposed to change anyway
- added routine for manipulating/editing the main site configuration (via ConfigAssistant)
- got rid of get_configuration(): we now use get_properties() for the main config too
- done with the area manager except 1 little detail (the handling of the data directory)
- added support for datatypes 'l' (picklist) and 'r' (radio) in ConfigAssistant; the type 'c' (checklist) still needs to be done

2009-03-14

- bumped version internal number to 2009031400 because of significant changes in the database (the extended configuration table format)

- added begin of support for 'Cascading Cascading Style Sheets' or 'Bazaar Style Style Sheets'.
- added yet more demodata to show the possibilities of the area manager

2009-03-06

- almost done with area manager; we only need to copy the theme parameters when adding a new area
- rearranged/renamed some phrases in order to keep at least a little overview

2009-03-05

- added a configassistant class as a quick tool for editing generic configuration data (first used in editing theme properties in areamanager)

2009-02-27

- moved all code dealing with area management to seperate class

2009-02-26

- added functionality to delete areas
- started with functionality to add areas

2009-02-25

- added fields mtime, muser_id, ctime, cuser_id to areas table
- bumped internal version to 2009022500
- bumped external version to 0.0.4
- added copy of webmaster account to account 'textmaster' to demodata: easy checking text interface with permissions of webmaster
- optimising queries on 'nodes' table by adding an index on field 'area_id'

2009-02-24

- (finally) started with configuration manager
- added has_site_permissions() to user class so we can use access control in area manager
- moved some code from pagemanagerlib to waslib so we can reuse it

2008-12-14

- incorporated a stub with base.css so we can see what a page will look like (more or less)
- minor cosmetics in the demo data

2008-12-10

- more navigation in base theme: now tree-like menu
- added language files en and nl to frugal theme
- minor cosmetics

2008-12-08

- added table themes_areas_properties
- added more demo data, including a quicklinks top + bottom in hidden sections
- changes in the built-in theme object
- bumped internal version to 2008120800
- added some more functionality to the built-in theme object

2008-12-04

- removed global \$NODE and \$AREA; we use a local variable \$area in main_index() and information about the node is to be found in the local variable \$tree
- obsoleted file area.class.php
- obsoleted file modulelib.php

- obsoleted file module.class.php

2008-12-03

- moved some more code from pagemanagerlib.php to waslib.php
- we no 'fix' circular references in function is_under_embargo()
(this should be fixed and properly done in a database repair tool or something)

2008-12-01

- moved build_tree() to waslib because it is used from main_index()
too and not just from main_admin()

2008-11-26

- some files re-arranged and renamed: e.g. /program/lib/adminlib.php
now lives in /program/main_admin.php
- added stubs to /cron.php (which call /program/main_cron.php)
and /index.php (which call /program/main_index.php)
- removed adminlib.php from CVS repository because it is now obsolete
- added begin of support for Apache's PATH_INFO feature for specifying
area and node like index.php/aaa/nnn or index.php/nnn. This does
pose a few problems with relative urls... to be fixed

2008-11-25

- yet another change to the database: modules table
- version bumped to 2008112500

2008-11-24

- added more logic to the garbage collection and expiry of obsolete
sessions
- added unconditional garbage collection after successful login
- added a default value for session_expiry to demodata
- renamed global config option sessionname to session_name throughout
- version bumped to 2008112400 because of change in demodata

2008-11-21

- first attempt to actually save htmlpage content
- version bumped to 20081121 because of change in data definition

2008-11-20

- added three stubs for modules: htmlpage, mypage and guestbook
- we now actually delegate the task of editing content to the module
admin code

2008-11-17

- we now take readonly into account when deleting nodes
- documentation in dialoglib
- minor cosmetics

2008-11-10

- save advanced node properties now working (except moving a non-empty
subtree to another area)
- bumped internal version number (WAS_VERSION) from 2008020100 to
2008111000. This forces an error message/condition code 050, which
means that the database is not in sync with the program. Normally,
this would mean that an updater should be executed which changes
the database structure. In the development phase we simple need to
reload the demodata. Naturally I changed the version number in the
demodata too.
- I also bumped the external version number from 0.0.1 to 0.0.2, just

to gain some experience with changing versions.

- Added a FAQ about condition code 050.
- Added a field 'locked_since' to the nodes table, allowing to make the 'sorry, record is locked' message even more friendly

2008-11-09

- minor changes:
 - . default visibility for a new page is now 'hidden' as per Dirk's request
 - . field order in basic node properties now has module as last in the list
 - . a 'default node' now loses the default node bit when the node is moved to another parent (preventing ending up with two defaults).
- added a class 'current' to options in the menu (currently there are two menus: the area menu and the node edit menu). The current item is underlined (via CSS).

2008-11-08

- we now manipulate the sort order in such a way that the sort order of other nodes is increased/decreased in order to make room for the moved node.

2008-11-07

- added stubs for module_connect() and module_disconnect() so we can actually link a module to a node (and unlink too, obviously)
- we can now almost save the data from the basic properties dialog (everything except the sort order by itself)

2008-11-06

- we can now fill the dialogs for editing a node with data from database
- added Dutch translations for all prompts etc.

2008-11-05

- rearranging the translations in /program/languages/em/admin.php
- minor change in admin_basic.css so we can indent options in a drop down list (tree)
- added a comment feature to the English translation file admin.php: this allows for communicating the purpose and context of strings to the translators
- completed the dialogs for basic and advanced editing of nodes (but not yet the actual saving etc.)

2008-11-04

- removed PERMISSION_NODE_EDIT_CHILD from useraccount class, changed into PERMISSION_NODE_EDIT_NODE because that makes more sense, I think.
- started with edit a node dialogs
- cosmetics

2008-10-31

- cleaning up code (it takes time to condense rough code and make it more elegant)
- added better logging to the alert feature
- on special request: now new nodes are added at the top of their section rather than at the end (some logic is more logic than other logic)
- we play a special trick when adding a node: we force switch to custom view with all ancestors opened and any other sections closed.

2008-10-30

- added a routine that churns out alert messages every now and then
- added more logging + alert to change default routine
- minor cosmetics

2008-10-29

- we now can add new sections and pages
- we now accumulate alerts when a new page/section is added

2008-10-28

- a small diversion in the past week: add tables for alert function (alerts, alerts_areas_nodes)
- updated demodata to 'play' with alerts
- added function to lock records of nodes table, alerts table

2008-10-20

- added even more functionality to dialoglib.php + documentation
- first attempt to do a dialog in pagemanager (no save yet) with new lib

2008-10-18

- added more functionality to dialoglib.php

2008-10-15

- added dialoglib.php

2008-10-13

- moved some html-utilities to new file /program/lib/htmllib.php
- added some links + translations to the start center
- first start on 'add a section' code

2008-10-10

- re-arranged icons, added new icons for visible/invisible page
- modified page preview to use a one-time access code via md5-hash
- documented all routines in pagemanagerlib.php, code cleanup
- implemented the 'set home' function for nodes

2008-10-09

- changed layout of node tree a little bit:
 - . add a node is now displayed at the top of the tree
 - . we no longer show sort_order and node_id in the anchor text (we do show the node_id in the mouse-over (title) though)
- added logic to switch between collapsed, customised, expanded tree view
- we now have stubs in place for all tasks in pagemanager, including preview
- we can now use index.php to show a preview (stub)

2008-10-08

- many changes in page manager: we now have an almost working tree view that can expand and collapse

2008-10-01

- inserted a new permission: PERMISSION_ADMIN. If this bit is set in the user's permissions (either site, area or node), the user has at least one permission bit set to manipulate the site, area or node. It is more or less shorthand for 'one of the permission bits is set'.
- we now have an area menu in pagemanager

2008-09-30

- a few tweaks in adminlib.php (http-equiv headers)
- added a help topic in help button in navigation in adminlib.php

2008-09-29

- moved workhorse /program/admin.php to /program/lib/adminlib.php to avoid confusion with file names (which 'admin.php' do you mean?)
- further documented adminlib.php, tweaked the page layout

2008-09-26

- basic dispatcher in admin.php done
- change in datadefinition: added field job_permissions to user table
- change in datadefinition: added field high_visibility to user table
- added support for high visibility via additional stylesheet
- basic admin.php navigation is more or less done

2008-09-22

- further refined page layout for admin.php
- added a basic version checker at <http://siteatschool.org/version/index.php>

2008-09-21

- started with admin.php

2008-06-10

- added shortcuts for \$CFG->www and \$CFG->progwww

2008-05-28

- changed \$NODE, \$AREA and \$THEME into objects
- added support for redirecting users if they accidentally request a page with an external link
- made a start with a base class for themes
- started with a minimalistic theme called 'frugal'
- created start of a theme factory
- minor cosmetics

2008-05-09

- changed the creation of the \$DB global object: we now use a database factory

2008-04-29

- removed some of the trigger_error() calls in waslib/calculate_node(); only database errors now emit to screen if debug is TRUE; the rest now use logger(...,LOG_DEBUG) which writes to the database in itself.
- we now kick unauthorised users/passersby out of protected areas with a fatal error 070 (cannot find node in area). They should login the regular way IMHO. I re-used error code '070' on purpose in order not to leak information about protected areas.

2008-04-25

- changed name of config parameter for language from 'language' to 'language_key' to stay in sync with other (database) fields dealing with language codes.
- moved all code dealing with translations and languages to a separate file and into a special class.

2008-04-23

- cleaned up loginlib.php, added more documentation
- added logger() which logs important events in the database

2008-04-21

- done with t() and supporting routines
- added en and nl translations for loginlib.php

2008-04-16

- busy with translating strings via function t()

2008-04-11

- changed interface for last_insert_id() method: use table+field instead of sequencename
- added new table login_failures
- loginlib.php now also implements the blacklist feature and the failures counters

2008-04-09

- login mostly works, except blacklisting/failure counts
- added confirmation mail to user after succesful change of password

2008-04-08

- added a javascript alert in every login dialog (near the end)
- created a random password/laissez passer-generator

2008-04-07

- changed password requirements in loginlib: now min 6 chars etc.
- rearranged code in loginlib
- added more demodata to play with

2008-04-05

- first draft of WAS-logo incorporated in login dialog

2008-04-04

- still working on login/logout and various failure modes and also on the 2-step laissez-passer + bypass routine

2008-04-03

- still working on login/logout and various failure modes

2008-04-02

- started with login/logout routines, added file loginlib.php

2008-04-01

- rearranged some code
- we now always read the complete config table in core
- added more config parameters in demodata
- refined session handling code
- added if_exist_session() type of routine to make sure we are not fooled by spurious cookies

2008-03-30

- added tabledef for users, sessions
- added more support routines for database manipulation to waslib.php
- added a new file dbsessionlib.php

2008-03-19

- renamed db_quoted() to db_escape_and_quote() to make the effect more obvious
- we now know the \$node and the \$area and even the \$theme to use
- re-arranged some code; moved to waslib.php

- added even more demodata to 'play' with index.php and we added a clickable breadcrumb trail (that was `_easy_`).

2008-03-17

- renamed the `quote()` method to `escape()` in Database class because it may add confusion; the difference between escaping apostrofs and adding apostrofs around a string in order to insert into the database like in `db_quote()`.
- added a simple test-routine to dump data from the database

2008-03-14

- added optional parameters `$limit` and `$offset` to `Database->query()` method
- we are now able to determine a valid area based on the user's request of 'area' and 'node'.

2008-03-13

- some more database manipulation routines added

2008-03-12

- evade the magic in `magic_quotes_sybase` by issuing a fatal error, circumvent the magic in `magic_quotes_gpc()` with function `magic_unquote()`
- changed the format of demodata from SQL-statements to nested arrays

2008-03-03

- added version check in `init.php`: the database version MUST match the code version, or else...

2008-02-29

- started with `demodata.php` to play with

2008-02-20

- added a simple `tabledefs` dumper to `install.php` so we can see `tabledefs` in a tabluar form
- more `tabledefinitions` in `tabledefs.php`

2008-02-19

- added much to the main `tabledefs.php`

2008-02-16

- added a subtree under `/program/install` for `tabledefs` of the main system and also places for storing language specific demo data
- started with main `tabledefs`
- quick and dirty testcode in `install.php` to excercise/test the `tabledefs`

2008-02-15

- moved a lot of test code from `install.php` to `/devel/test` directory so I can 'play' with various subsystems without interfering with the real code. Currently `/deve/test/test.php` is focused on excercising the Database class.

2008-02-14

- added an extra dummy parameter to `$DB->last_insert_id()` as hook for future extension with other database drivers
- added a function `$DB->table_exists()`
- added a test with some 128 `fielddefs` in `install.php` (for now)

2008-02-12

- added `$DB->debug` (default `FALSE`), can be set via constructor

- added \$DB->drop_table()
- added \$DB->create_table() using 'generic' (not MySQL-specific) datadefinition
- added another file to 'play' with: /program/install.php
- moved diff_microtime() to init.php

2008-02-11

- added new object type DatabaseResult to deal with queries returning data
- added a few more stubs in index.php to casually test Database/DatabaseResult-classes

2008-02-10

- documented \$CFG->debug
- got a few more methods working in mysql.class.php + q&d tests in index.php

2008-02-07

- experiments with the 'poor mans' database factory were more or less successful: the code works, but phpdoc fails badly when the same class is defined multiple times within the same package. However, a postgresql-driver in a separate @packages does work with phpdoc. Oh well.
- re-arranged some code in the Database-class: we now have a separate method to connect to the server and open the database, we no longer do that from the constructor. Also, we no longer bail out on MySQL-errors because we can expect errors during install and we don't want to bail out at that point.
- added error and errno to Database class which store the results from the underlying database, handy to find out what, exactly, went wrong.
- added the parameter \$CFG-debug in init.php
- moved the unset(\$CFG) from config(-example).php to index.php in order to make certain that we won't be fooled by a stray global set via \$_GET or whatever user input
- added a fake microtime() function in case it doesn't exist in the environment. The fake version uses a fractional part of 0.0 but is otherwise 100% compatible with the real microtime() string format. The reported time will be off by at most 2 seconds, oh well.
- Added a global object to record the performance, including the time of start and stop of the script execution. Could be used to identify bottlenecks.
- rearranged the order of code in init.php
- added error_reporting depending on the value of \$CFG->debug.
- brought the call to the new Database class in line with the new class definition
- removed the calls to error_exit() from the Database class; they do not belong there.
- documented the format of the release date variable in version.php to include the full seconds-resolution ISO 8601 date
- added more error checking in Database class via \$DB->errno and \$DB->error

2008-02-06

- made a start with databaselib.php and mysql.class.php
- some tweaking in version.php: now record full release date+time

2008-02-04

- re-arranged the comments in config-example.php, hopefully clarifies

2008-02-01

- added init.php
- added version.php: keeps track of version of php-files (for

comparison with database to determine necessary update actions)

2008-01-31

- added preliminary versions of README, INSTALL, etc.
- completed a quick and dirty script to generate documentation with phpdoc
- completed a script to generate both distribution version and development version of W@S

2008-01-28

- committed a first version of /index.php and config-example.php to CVS; we have now actually started creating program code

CREDITS

/program/CREDITS.txt

\$Id: CREDITS.txt,v 1.1.1.1 2011-02-01 13:00:01 pfokker Exp \$

Website@School Team

=====

Barry Faas - graphics designer
Dirk Schouten - project leader
Karin Abma - project leader
Peter Fokker - main developer

Translators

=====

...

Contributors

=====

...

Sponsors

=====

They donated money or equipment to the Website@School Project:

...

Appendix D - Todo List

In Package wascore

In [AclManager](#)

- there is something not right with buffering the tabledefs. If an error occurs, we get FALSE instead of an array. Mmmmm....

In [GroupManager::acl_delete\(\)](#)

- should this routine be moved to an acl-object? Hmmm....

In [AdminOutput::AdminOutput\(\)](#)

- do we need a link rel="shortcut icon" type of header too?
- do we really need more meta-headers?
- is it really wise to add a base header? It interferes with the session cookie whenever you login at another URL than the base+'admin.php'... Comment it out for now

In [AdminOutput](#)

- add a 'funnel mode': disable all distracting links that could seduce the user to leave and leave locked records (eg. nodes)
- carefully check if we need more headers in html-head section of document, see AdminOutput().

In [Area](#):

- refactor/change the way the default node and area are calculated from the requested node and

area. We now go the the database too often.

In [area.class.php](#)

- we probably need to get rid of this file because it is not used (2010-12-07/PF)

In [AreaManager](#)

- we need to take care of spurious spaces in inputs (or do we?)

In [AreaManager::area_delete\(\)](#)

- should we also require the user to delete any files associated with the area before we even consider deleting it? Or is OK to leave the files and still delete the area. We do require that nodes are removed from the area, but that is mainly because of maintaining referential integrity. Mmmmm... Maybe that applies to the files as well, especially in a private area. Food for thought.
- since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

In [AreaManager::area_overview\(\)](#)

- should we add a paging function to the list of areas? Currently all areas are shown in a single list...
- should we make two categories: 'public' and 'private' in the list of areas? Maybe handy when there are many manu areas, but it would be inconsistend with the page manager menu which simply lists the areas in the sort order. Easy way out: the user is perfectly capable to set the sort order in such a way that the sort order already groups the public and private areas. Oh well....

In [AreaManager::area_setdefault\(\)](#)

- should we acknowledge the changed default to the user or is it enough to see the icon 'move'?
- should we send alerts? If so, can we use the routine to queue messages from pagemanager? A reason not to send alerts: the alerts will be sent as soon as a page is added to the new area,

so why bother?

In [build_tree\(\)](#)

- repairing a node doesn't really belong here, in this routine. we really should have a separate 'database repair tool' for this purpose. someday we'll fix this....
- what if we need the trees of two different areas? should the static var here be an array, keyed by area_id?

In [Node::calculate_node\(\)](#)

- refactor into two different functions: one for specified node, other for unspecified node

In [Theme::calc_breadcrumb_trail\(\)](#)

- split into two separate routines, one to set the tree, another to construct the list of anchors

In [Theme::calc_tree_visibility\(\)](#)

- how about making all nodes under embargo visible when previewing a page or at least the path from the node to display?

In [TranslateTool::code_highlight\(\)](#)

- should we turn to ereg() instead of a simple str_replace() for {VARIABLE} highlighting?

In [DatabaseMysql::column_definition\(\)](#)

- 'enum' type equivalent with varchar, enum_values[] array is not used at all, only as a form of documentation
- should we allow both int and integer?

In [DatabaseMysql::concat\(\)](#)

- perhaps extend this function to accept more than 2 strings?

In [ConfigAssistant](#)

- implement checklist

In [DatabaseMysql::connect\(\)](#)

- weigh pros and cons of persistent database connections, perhaps add as config option?

In [convert_to_type\(\)](#)

- perhaps change the possible values of \$type to full strings rather than 'cryptic' single letter codes. Furthermore: what do we do with invalid dates, times and date/times? For now it is a stub, returning \$value as-is. Oh well.

In [DatabaseMysql::create_table\(\)](#)

- document correct link for documentation of generic table definition 'tabledefs.php'

In [DatabaseMysql::create_table_sql\(\)](#)

- document correct link for documentation of generic table definition 'tabledefs.php'
- find a way to deal with the enum values: where do we keep them? Or do we keep them at all?

In [database_factory\(\)](#)

- perhaps add postgresql in a future version

In [dbsession_close\(\)](#)

- should we do something with locking the session record from `dbsession_open()` until `dbsession_close()`? For now, the session record is not locked in any way, so the latest call gets to keep its changes Mmmm....

In [dbsession_create\(\)](#)

- should we also record the IP-address of the user in the session record? In a way this is a case of information leak, even though it is only between authenticated users. Mmmm...

In [GroupManager::delete_group_capacities_acls\(\)](#)

- should we also require the user to delete any files associated with the area before we even consider deleting it? Or is it OK to leave the files and still delete the area. We do require that nodes are removed from the area, but that is mainly because of maintaining referential integrity. Mmmmm... Maybe that applies to the files as well, especially in a private area. Food for thought.
- since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

In [dialog_get_label\(\)](#)

- if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

In [dialog_get_widget\(\)](#)

- we could manipulate the title attribute of input strings like "please enter a number between {MIN} and {MAX}" based on the various value properties instead of just displaying the title. oh well, for a future version, perhaps...
- we now only cater for buttons via input type="submit" without the option to visualise the accesskey. Using the button tag could solve that, but button is not defined before HTML 4.01. What to do?

In [dialog_get_widget_file\(\)](#)

- if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?
- should we do something with an um-empty \$value? If so, waht? The browser ignores this...

In [dialog_get_widget_richtextinput\(\)](#)

- if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

In [dialog_get_widget_textinput\(\)](#)

- if we let the hotkey from the label prevail and add it to the input tag, why add a hotkey to the label too?

In [AclManager::dialog_tableform\(\)](#)

- bailing out on non-array is a crude way of error handling: this needs to be fixed

In [error_exit\(\)](#)

- do we really want to 'leak' a link to the main site?

In [FileManager::FileManager\(\)](#)

- a nice filter for JOB_IMAGEBROWSER and also an alternative user interface for browsing/selecting images

In [Theme::get_address\(\)](#)

- should we add additional text to the address, e.g. prefix 'URL: ' or something? now it is just a plain old URL without any comments whatsoever.

In [ConfigAssistant::get_dialogdef\(\)](#)

- implement checklist

In [PageManager::get_dialogdef_add_node\(\)](#)

- should we make the defaults in this routine configurable? (I'm not convinced they should)

In [AclManager::get_dialogdef_admin\(\)](#)

- handle the related information in this dialog

In [AclManager::get_dialogdef_intranet\(\)](#)

- handle the related information in this dialog

In [TranslateTool::get_dialogdef_language_domain\(\)](#)

- try to figure this out: when the delimiter in \$name was a dot '.' \$_POST contained a '_' instead. WTF? (it seems that a colon works... for now)

In [Theme::get_div_breadcrumbs\(\)](#)

- how about adding a title to the items? or do we do that already?

In [UserManager::get_editor_names\(\)](#)

- this list here is hardcoded: we do not expect to be adding or removing editors to/from the CMS soon. However, it might be cleaner to do this elsewhere.

In [Theme::get_html_head\(\)](#)

- also deal with Bazaar Style Style Sheets at node level in this routine

In [PageManager::get_icon_delete\(\)](#)

- should we display trash can icons for sections with non-empty subsections? there really is no point, because we eventually will not accept deletion of sections with grandchildren in task_node_delete. Hmmmmmm..... For now I just added the condition that access is denied when a section has grandchildren. Need to refine this, later. Also, how about readonly nodes? Surely those cannot be deleted... should it not show in the icon?

In [AreaManager::get_icon_delete\(\)](#)

- should we check to see if the area is empty before showing delete icon? Or is it soon enough to refuse deletion when the user already clicked the icon? I'd say the latter. For now...

In [PageManager::get_icon_edit\(\)](#)

- move permission check to a separate function permission_edit_node()

In [PageManager::get_icon_page_preview\(\)](#)

- if this is a public area, the user can see every page, except the expired/embargo'ed ones should we take that into account too? I'd say that is way over the top. How about pages in an intranet where the user has view privilege? Complicated. KISS: only show preview to those that can edit or edit content.

In [Theme::get_logo\(\)](#)

- should we take path_info into account here too???? how about /area/aaa/node/nnn instead of

/aaa/hnn???

In [get_mimetype\(\)](#)

- there is room for improvement here: the code in step 1 and step 2 is largely untested

In [AdminOutput::get_navigation\(\)](#)

- we need to clean up this code and properly implement the funnel mode (2009-12-18)

In [Language::get_phrase\(\)](#)

- should we return an error for an invalid specific language?

In [Theme::get_quicklinks\(\)](#)

- should we take Apache's PATH_INFO feature into account to create friendly links?

In [GroupManager](#)

- Perhaps this class should be merged with the UserManager class because there is a lot of overlap. Mmmmm.... maybe in a future refactoring operation.

In [GroupManager::group_delete\(\)](#)

- should we also require the user to delete any files associated with the group before we even consider deleting it? Or is it OK to leave the files and still delete the group. Food for thought.
- since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

In [GroupManager::group_savenew\(\)](#)

- maybe we should find a more elegant way to check a field for uniqueness
- should we delete the datadirectory if something goes wrong?

In [href\(\)](#)

- should we merge this with `html_a()` and/or rename this routine to `html_href()`?

In [is_expired\(\)](#)

- this function also 'repairs' circular references. This should move to a separate function but for the time being it is "convenient" to have automatic repairs... tree-repair

In [is_under_embargo\(\)](#)

- this function also 'repairs' circular references. This should move to a separate function but for the time being it is "convenient" to have automatic repairs... tree-repair

In [job_start\(\)](#)

- this routine is a stub

In [job_tools\(\)](#)

- fix permissions for backup tool! perhaps another bit?

In [TranslateTool::languages_overview\(\)](#)

- should we add a paging function to the (perhaps looooong) list of languages?

In [lock_record\(\)](#)

- do we need a 'force lock' option to forcefully take over spurious locks?
- perhaps we can save 1 trip to the database by checking for something like `UPDATE SET locked_by = $session_id WHERE (id = $id) AND ((locked_by IS NULL) OR (locked_by = $session_id))` but I don't know how many affected rows that would yield if we already had the lock and effectively nothing changes in the record. (Perhaps always update atime to force 1 affected row?)
- we need to resolve the problem of crashing browsers and locked records

In [logger\(\)](#)

- should we make this configurable and maybe log directly to syslog (with automatic logrotate) or do we want to keep this 'self-contained' (the webmaster can read the table, but not the machine's syslog)?

In [loginlib.php](#)

- should we suppress the username in the laissez-passer routine? We do leak the the username in an insecure email message. This does require making the laissez-passer code unique in the database (currently only username+code has to be unique and that's easy because the username itself is unique).

In [login_dialog_open\(\)](#)

- should we add another 'powered by' link to '/program/about.html'?

In [login_stylesheet\(\)](#)

- this routine needs some cleaning up

In [main_admin\(\)](#)

- should we cater for a special 'print' button + support for a special style sheet for

media="print"?

In [main_file\(\)](#)

- the check on '/../' is inconclusive if the \$path is encoded in UTF-8: the overlong sequence 2F C0 AE 2E 2F eventually yields 2F 2E 2E 2F or '/../'. Reference: RFC3629 section 10.

In [main_index\(\)](#)

- cleanup login/logout-code

In [main_index.php](#)

- add the performance results in a HTML-comment if not CFG->debug, in sight otherwise

In [manual.php](#)

- guess what? we need to replace this stub with real documentation
- How about adding an extra parameter to manual.php in order to 'deep link' into the manual?

In [module.class.php](#)

- we probably need to get rid of this file because it is not used (2010-12-07/PF)

In [modulelib.php](#)

- we probably need to get rid of this file because it is not used (2010-12-07/PF)

In [PageManager::module_connect\(\)](#)

- should we pass the `area_id` at all? What happens when a node is moved to another area without informing the module? Questions, questions, questions...

In [PageManager::module_disconnect\(\)](#)

- should we pass the `area_id` at all? What happens when a node is moved to another area without informing the module? Questions, questions, questions...

In [module_factory\(\)](#)

- what if the module is not found? Currently no alternative is loaded but `FALSE` is returned.

In [PageManager::module_load_admin\(\)](#)

- should we sanitise the `modulename` here? It is not user input, but it comes from the modules table in the database. However, if a module name would contain sequences of `"../"` we might be in trouble

In [module_load_view\(\)](#)

- should we sanitise the `modulename` here? It is not user input, but it comes from the modules table in the database. However, if a module name would contain sequences of `"../"` we might be in trouble

In [Node:](#)

- we probably need to get rid of this file because it is not used (2010-12-07/PF)

In [performance_get_seconds\(\)](#)

- maybe we should get rid of this `$PERFORMANCE` object, because it doesn't do that much anyway

In [PageManager::permission_delete_node\(\)](#)

- we should also take the readonly flag into account (or should we?) when determining delete permissions

In [quoted_printable\(\)](#)

- should we change the code to accomodate the canonical newline CRLF in the input?

In [Email::rfc2047_qstring\(\)](#)

- maybe optimise this routine to let pure ASCII-words through unencoded (in a later version)

In [sanitise_filename\(\)](#)

- should we check for overlong UTF-8 encodings: C0 AF C0 AE C0 AE C0 AF equates to /../ or is that dealt with already by filtering on letters/digits and embedded dots/dashes/underscores?

In [AclManager::save_data_admin\(\)](#)

- fix the crude error check on dialogdef === FALSE here

In [PageManager::save_node\(\)](#)

- there is something wrong with embargo: should we check starting at parent or at node? this is not clear: it depends on basic/advanced and whether the embargo field changed. mmmm... safe choice: start at node_id for the time being
- this routine could be improved by refactoring it; it is too long!

In [Email::set_header\(\)](#)

- should we bring the Capitalisation of \$name in line with the default capitalisation in the list above?

In [FileManager::show_directories_and_files\(\)](#)

- This routine is way too long, it should be split up into smaller subroutines

In [FileManager::sort_entries\(\)](#)

- it is a pity I cannot reference \$this->sort from within the 6 cmp-functions...

In [task_logview\(\)](#)

- should we allow for fancy selection mechanisms on the logfile or is that over the top?

In [PageManager::task_node_delete\(\)](#)

- should we display trash can icons for sections with non-empty subsections in treeview? there really is no point, because we eventually will not accept deletion of sections with grandchildren. Hmmmmm.....

In [PageManager::task_page_preview\(\)](#)

- the check on permissions can be improved (is PERMISSION_XXXX_EDIT_NODE enough?)
- there is an issue with redirecting to another site: officially the url should be fully qualified (ie. \$CFG->www). I use the shorthand, possibly without scheme and hostname (\$CFG->www_short). This might pose a problem with picky browsers. See [calculate uri shortcuts](#) for more information.

In [PageManager::task_save_newnode\(\)](#)

- about 'sort_order': do we insert nodes at the end or the beginning of a parent section?
- how do we alert users that an embargo date has come around? Do we schedule alerts via cron?

In [theme_factory\(\)](#)

- should we massage the directory and file names of the included theme?
- what if the theme is not found? Currently no alternative is loaded but FALSE is returned.

In [FileManager::unique_filename\(\)](#)

- Should we take care of the race condition in this routine? Should we already create an empty file or is that clutter?

In [update_statistics\(\)](#)

- maybe extend this routine to actually store more statistics information in a separate table

In [UserManager](#)

- Perhaps this class should be merged with the GroupManager class because there is a lot of overlap. Mmmmm.... maybe in a future refactoring operation.

In [UserManager::user_delete\(\)](#)

- should we also require the user to delete any files associated with the user before we even consider deleting it? Or is it OK to leave the files and still delete the user. Food for thought.
- since multiple tables are involved, shouldn't we use transaction/rollback/commit? Q: How well is MySQL suited for transactions? A: Mmmmm.... Which version? Which storage engine?

In [UserManager::user_edit\(\)](#)

- maybe it is better to call this routine with \$user_id as a parameter? that allows for moving from adduser() -> saveuser() -> edituser(\$user_id). Mmmm, food for thought

In [UserManager::user_savenew\(\)](#)

- maybe we should find a more elegant way to check a field for uniqueness
- shouldn't we end with the edit-user dialog rather than the users overview? that might make more sense...

In [FileManager::valid_path\(\)](#)

- the check on '/../' is inconclusive if the \$path is encoded in UTF-8: the overlong sequence 2F C0 AE 2E 2F eventually yields 2F 2E 2E 2F or '/../'. Reference: RFC3629 section 10.

In [FileManager::virusscan\(\)](#)

- maybe use MIME for sending alert if not 7bit message?
- This routine is quite *nix-centric. I'm not sure how this would work other server platforms. Should we do something about that?

In [Zip::zip_add_data\(\)](#)

- should we handle the option of a better compression level (eg. level 9) in gzcompress()? we could check to see if CMF equals 0x78 and FLG is either 0x01, 0x5E, 0x9C or 0xDA the latter 4 values might have an effect on general purpose bit flag bits 2 and 3. for now we'll just keep it simple, but there might be a little something to improve here.
- should we handle the possibility of an additional 4 bytes for DICTID (RFC1950, reference [2])?

In Package wasinstall

In [InstallWizard::check_compatibility\(\)](#)

- add more tests, e.g. for gd, safe_mode, memory limit, etc.

In [demodata_users_groups\(\)](#)

- get rid of the \$wizard kludge!
- should we append an underscore to the userpaths to make sure we don't clash with the first user account?

In [InstallWizard::get_default_install_values\(\)](#)

- should we check the program version versus the stored program version here?
- there is something wrong with the default for \$cms_www; FIXME (commented out for now)

In [InstallWizard::get_page\(\)](#)

- should we promote language and high_visibility to function parameters instead of using \$_SESSION directly?

In [install.php](#)

- how prevent third party-access to install.php after initial install? .htaccess? !exists(../config.php)?
- we should make sure that autosession is disabled in php.ini, otherwise was won't work
- we should make sure that register globals is off
- we should make sure that we can actually set cookies (necessary when logging in).

In [InstallWizard::perform_installation\(\)](#)

- should we save the config.php to the datadir if the main dir fails? Mmmm.... security implications?
- this routine badly needs refactoring

In [InstallWizard::save_cms\(\)](#)

- also take `safe_mode` into account? Should that be a requirement for successful installation?

In [InstallWizard::show_dialog_cms\(\)](#)

- can we suppress even more fields here in case of a Standard installation?

In [InstallWizard::show_dialog_compatibility\(\)](#)

- more tests to perform here: safe mode, memory limit, processing time limit, register globals

In [tabledefs.php](#)

- automatically create appropriate sequence name for serial fields??? or add seqdefs too?

In [InstallWizard::write_config_php\(\)](#)

- should we make the filemode (hardcoded at 0400) configurable/customisable?

In Package wasmod_htmlpage

In [htmlpage_cron.php](#)

- change this stub into a real cron function.

In [htmlpage_search.php](#)

- change this stub into a real search function, with limits on the number of results, an offset where to start and perhaps even a time limit. for now this always returns an empty array

Index

A

AdminOutput::get_lines()	186
<i>get lines from an array in a single properly indented string</i>	
AdminOutput::get_html_head()	186
<i>get all lines in the HTML head section in a single properly indented string</i>	
AdminOutput::get_logo()	187
<i>construct an image tag with the W</i>	
AdminOutput::get_menu()	187
<i>get all lines in the menu DIV in a single properly indented string</i>	
AdminOutput::get_navigation()	187
<i>construct a navigation bar for various jobs the user is allowed to do</i>	
AdminOutput::get_html()	186
<i>construct an output page in HTML</i>	
AdminOutput::get_div_messages()	185
<i>get a perhaps bulleted list of messages in a DIV</i>	
AdminOutput::get_address()	184
<i>return the reconstructed URL in a single (indented) line</i>	
AdminOutput::add_stylesheet()	184
<i>add a link to a stylesheet to the HTML head part of the document</i>	
AdminOutput::get_bottomline()	184
<i>report basic performance indicators in a single line</i>	
AdminOutput::get_breadcrumbs()	185
<i>retrieve/construct a list of 0 or more clickable breadcrumbs</i>	
AdminOutput::get_content()	185
<i>get all lines in the content DIV in a single properly indented string</i>	
AdminOutput::get_pagination()	188
<i>retrieve/construct a list of 0 or more clickable links to paginated screens</i>	
AdminOutput::get_popups()	188
<i>construct javascript alerts for messages</i>	
Area	191
<i>Methods to access properties of an area</i>	
AdminOutput::set_suppress_output()	190
<i>manipulate output suppression</i>	
Area::\$area_exists	191
Area::\$area_id	191
Area::\$area_record	191
AdminOutput::set_helptopic()	190
<i>set the additional help topic to show when user clicks help button</i>	
AdminOutput::set_funnel_mode()	190
<i>manipulate the funnel mode</i>	
AdminOutput::get_quickbottom()	189
<i>construct a list of quicklinks for botton of page</i>	
AdminOutput::get_quicktop()	189
<i>construct a list of quicklinks for top of page, including logout link</i>	
AdminOutput::send_headers()	190

<i>send collected HTTP-headers to user's browser</i>	
AdminOutput::send_output()	190
<i>send collected output to user's browser</i>	
AdminOutput::add_popup_top()	184
<i>add a message to the list of popup-messages at the TOP of the document</i>	
AdminOutput::add_popup_bottom()	184
<i>add a message to the list of popup-messages at the BOTTOM of the document</i>	
AdminOutput::\$messages_bottom	179
AdminOutput::\$menu	178
AdminOutput::\$messages_inline	179
AdminOutput::\$messages_top	179
AdminOutput::\$pagination	179
AdminOutput::\$http_headers	178
AdminOutput::\$html_head	178
AdminOutput::\$dtd	177
AdminOutput::\$funnel_mode	177
AdminOutput::\$helptopic	178
AdminOutput::\$high_visibility	178
AdminOutput::\$subtitle	179
AdminOutput::\$suppress_output	180
AdminOutput::add_meta()	182
<i>add a line with meta-information to the HTML head part of the document</i>	
AdminOutput::add_message()	182
<i>add a message to the list of inline messages, part of the BODY of the document</i>	
AdminOutput::add_meta_http_equiv()	182
<i>add a line with http-equiv meta-information to the HTML head part of the document</i>	
AdminOutput::add_pagination()	182
<i>add a pagination navigation bar to the output</i>	
AdminOutput::add_pagination_item()	183
<i>add a link to screen of a paginated list to the existing list</i>	
AdminOutput::add_menu()	182
<i>add a line to the menu part of the document</i>	
AdminOutput::add_http_header()	181
<i>add an HTTP-header</i>	
AdminOutput::\$title	180
AdminOutput::add_breadcrumb()	181
<i>add a breadcrumb to the breadcrumb trail</i>	
AdminOutput::add_content()	181
<i>add a line or array of lines to the content part of the document</i>	
AdminOutput::add_html_header()	181
<i>add a header to the HTML head part of the document</i>	
Area::\$module	192
Area::\$nodes	192
AreaManager::area_savenew()	205
<i>save the newly added area to the database</i>	
AreaManager::area_save()	204
<i>validate and save modified data to database</i>	
AreaManager::area_setdefault()	205
<i>make the selected area the default for the site</i>	
AreaManager::a_param()	206
<i>shorthand for the anchor parameters that lead to the area manager</i>	
AreaManager::count_existing_theme_properties()	206
<i>determine the number of existing properties for a theme in an area</i>	

AreaManager::area_resettheme()	204
<i>reset the theme configuration to the factory defaults</i>	
AreaManager::area_overview()	203
<i>display list of areas with edit/delete icons etc. and option to add an area</i>	
AreaManager::area_add()	201
<i>present a dialog where the user can enter minimal properties for a new area</i>	
AreaManager::\$show_parent_menu	201
AreaManager::area_delete()	202
<i>delete an area from this site after confirmation</i>	
AreaManager::area_edit()	202
<i>show the basic properties edit dialog and the edit menu</i>	
AreaManager::area_edittheme()	203
<i>show the theme/area configuration dialog and the edit menu</i>	
AreaManager::get_dialogdef_add_area()	206
<i>construct the add area dialog</i>	
AreaManager::get_dialogdef_edit_area()	206
<i>construct the edit area basic properties dialog</i>	
AreaManager::show_dialog_confirm_delete()	209
<i>show the name of an area and ask the user for a confirmation of deletion</i>	
AreaManager::reset_theme_defaults()	209
<i>reset the theme properties of an area to the default values</i>	
AreaManager::show_edit_menu()	210
<i>display the edit menu via \$this->output</i>	
AreaManager::show_parent_menu()	210
<i>allow the caller to use the menu area (or not)</i>	
AreaManager::sort_order_new_area()	210
<i>determine the value for the sort order of a new area</i>	
AreaManager::get_theme_records()	209
<i>retrieve a list of all available theme records</i>	
AreaManager::get_options_themes()	208
<i>fetch a list of themes available for an area</i>	
AreaManager::get_dialog_data()	207
<i>fill the dialog with current area data from the database</i>	
AreaManager::get_icon_delete()	207
<i>construct a clickable icon to delete this area</i>	
AreaManager::get_icon_edit()	207
<i>construct a clickable icon to edit theme properties of this area (edit advanced)</i>	
AreaManager::get_icon_home()	208
<i>construct a clickable icon to set the default area</i>	
AreaManager::\$output	200
AreaManager::\$areas	200
Area::build_tree_of_nodes()	194
<i>construct a complete tree from node records (including 'hidden' nodes)</i>	
Area::area_is_private()	194
<i>determine if an area is private or public</i>	
Area::calculate_default_descendant_node_id()	195
<i>calculate the default page in the subtree \$subtree_id in area \$area_id</i>	
Area::calculate_node_id()	195
<i>determine which node in which area to show</i>	
Area::calculate_validate_default_node_id()	195
<i>calculate and validate the default node from an area or the default area</i>	
Area::\$tree	193
Area::\$table_nodes_prefix	193

Area::\$node_id	192
Area::\$requested_area	192
Area::\$requested_node	192
Area::\$table_areas_prefix	193
Area::calculate_validate_node_id()	196
<i>calculate and validate the node to display based on a node and an area or the default area</i>	
Area::exists()	196
<i>determine existence of area</i>	
Area::node2anchor()	198
<i>construct an anchor from a node record</i>	
Area::get_theme_id()	198
<i>determine the theme to use</i>	
Area::retrieve_nodes_from_database()	199
<i>get an array of all available node records in the selected area as assoc arrays</i>	
Area::subtree_is_hidden()	199
<i>recursively determine whether all children of a node are hidden</i>	
AreaManager	200
<i>Methods to access properties of an area</i>	
Area::get_node_title()	198
<i>fetch the title of a node</i>	
Area::get_node_record()	198
<i>get a node record, maybe from the cache</i>	
Area::get_area_title()	196
<i>fetch the title to be used in a HTML-title-tag in the head section</i>	
Area::get_breadcrumb_anchors()	197
<i>fetch breadcrumb trail for a node</i>	
Area::get_children()	197
<i>get an ordered array of node records with parent equal to \$node_id</i>	
Area::get_node_link_href()	197
<i>get the link_href property of a node</i>	
AdminOutput::\$content	177
AdminOutput::\$breadcrumbs	177
accesskey tilde to underline()	50
<i>replace tilde+character with emphasised character to indicate accesskey</i>	
accesskey from string()	49
<i>return the character that follows the first tilde in a string</i>	
acceptable_new_password()	78
<i>check the new passwords satisfy password requirements</i>	
authenticate_user()	79
<i>check the user's credentials in one of three ways</i>	
ACL_ROLE_GURU	111
ATTR_CLASS_VIEWONLY	49
ATTR_CLASS_ERROR	49
AREAMANAGER_DIALOG_DELETE	25
AREAMANAGER_DIALOG_ADD	25
AREAMANAGER_DIALOG_EDIT	25
AREAMANAGER_DIALOG_EDIT_THEME	25
AREAMANAGER_DIALOG_RESET_THEME	25
ACL_ROLE_INTRANET_ACCESS	111
ACL_ROLE_NONE	111
add_javascript_select_url_function()	137
<i>add javascript code that implements a url selection (used in integration with FCKeditor)</i>	
add_javascript_popup_function()	137

<i>add javascript code that implements a popup to the header part of the page</i>	
admin_continue_session()	137
<i>continue the session from the previous call OR exit</i>	
admin_login()	138
<i>perform a step in the login procedure</i>	
admin_logout_and_exit()	138
<i>logout the user and exit</i>	
appropriate_legal_notices()	115
<i>construct a link to appropriate legal notices as per AGPLv3 section 5</i>	
ACL_ROLE_PAGEMANAGER_SITEMASTER	111
ACL_ROLE_PAGEMANAGER_AREAMASTER	111
ACL_ROLE_PAGEMANAGER_CONTENTMASTER	111
ACL_ROLE_PAGEMANAGER_PAGEMASTER	111
ACL_ROLE_PAGEMANAGER_SECTIONMASTER	111
AREAMANAGER_CHORE_VIEW	25
AREAMANAGER_CHORE_SET_DEFAULT	25
ACL_LEVEL_SECTION	22
<i>limit available role options to pages and sections (used in pagemanager permissions)</i>	
ACL_LEVEL_PAGE	22
<i>limit available role options to pages (used in pagemanager permissions)</i>	
ACL_LEVEL_SITE	22
<i>no limit on available role options (used in pagemanager permissions)</i>	
ACL_TYPE_ADMIN	22
<i>acl for administrator permissions</i>	
ACL_TYPE_INTRANET	22
<i>acl for intranet permissions</i>	
ACL_LEVEL_NONE	22
<i>limit available role options to 'none' and 'guru' (used in pagemanager permissions)</i>	
ACL_LEVEL_AREA	22
<i>limit available role options to pages, sections and areas (used in pagemanager permissions)</i>	
admin.php	13
<i>/program/languages/en/admin.php - translated messages for /program/admin.php (English)</i>	
admin.php	16
<i>/program/languages/nl/admin.php - translated messages for /program/admin.php (Dutch)</i>	
accountmanagerlib.php	19
<i>/program/lib/accountmanagerlib.php - accountmanager (users and groups)</i>	
aclmanager.class.php	22
<i>/program/lib/aclmanager.class.php - dealing with access control lists</i>	
ACL_TYPE_MODULE	23
<i>acl for individual module permissions</i>	
ACL_TYPE_PAGEMANAGER	23
<i>acl for pagemanager permissions</i>	
AREAMANAGER_CHORE_EDIT_THEME	25
AREAMANAGER_CHORE_EDIT	25
AREAMANAGER_CHORE_RESET_THEME	25
AREAMANAGER_CHORE_SAVE	25
AREAMANAGER_CHORE_SAVE_NEW	25
AREAMANAGER_CHORE_DELETE	25
AREAMANAGER_CHORE_ADD	25
area.class.php	24
<i>/program/lib/area.class.php - taking care of areas</i>	
AREA_CONST_TABLE_AREAS	24
<i>the bare name of the areas table (without prefix), should be a class constant but PHP4 doesn't</i>	

do that	24
AREA CONST TABLE NODES	24
the bare name of the nodes table (without prefix), should be a class constant but PHP4 doesn't do that	
areamanager.class.php	25
/program/lib/areamanager.class.php - taking care of area management	
admin_show_login_and_exit()	139
show login dialog and exit	
AclManager	156
class for manipulating (edit+save) access control lists	
AclManager::get_roles_pagemanager()	171
construct an option list with roles for pagemanager access	
AclManager::get_roles_intranet()	171
contstruct an option list with roles for intranet access	
AclManager::save_data()	172
save the changed data for the selected acl_type	
AclManager::save_data_admin()	172
save changed job permissions to the database	
AclManager::save_data_intranet()	172
save the changed roles for intranet access to the tables 'acls' and 'acls_areas'	
AclManager::get_permissions_nodes_in_area()	171
retrieve an array with 0, 1 or more records with permissions from table 'acls_nodes'	
AclManager::get_permissions_areas()	170
retrieve an array with 0, 1 or more records with permissions from table 'acls_areas'	
AclManager::get_dialogdef_pagemanager()	169
construct a dialog definition for pagemanager permissions	
AclManager::get_icon_area()	169
construct a clickable icon to open/close this area	
AclManager::get_icon_blank()	170
construct a spacer of standard icon width (to line up items)	
AclManager::get_permissions()	170
retrieve an array with 0, 1 or more records with permissions from table 'acls'	
AclManager::save_data_pagemanager()	172
save the changed roles for pagemanager to the tables 'acls' and 'acls_areas' and 'acls_nodes'	
AclManager::save_data_permissions()	172
save the changed roles in the dialog to the corresponding tables 'acls'	
AclManager::show_dialog_intranet()	174
display a tabular form for manipulating intranet permissions	
AclManager::show_dialog_admin()	174
display a tabular form for manipulating admin permissions	
AclManager::show_dialog_pagemanager()	175
display a tabular form for manipulating pagemanager permissions	
AclManager::show_tree_walk()	175
add the specified node to dialogdef, optionally all subtrees, and subsequently all siblings	
AdminOutput	176
conveniently collect output	
AclManager::show_dialog()	174
show the dialog where the selected Acl can be modified	
AclManager::set_related_acls()	174
further initialise the AclManager with related Acl's	
AclManager::set_action()	173
further initialise the AclManager with the dialog action property	
AclManager::set_dialog()	173

<i>further initialise the AclManager with the dialog identification</i>	173
AclManager::set_header()	173
<i>further initialise the AclManager with the dialog header</i>	
AclManager::set_intro()	173
<i>further initialise the AclManager with the dialog introductory text</i>	
AclManager::get_dialogdef_intranet()	168
<i>construct an array with the intranet dialog information</i>	
AclManager::get_dialogdef_admin()	168
<i>construct an array with the admin dialog information</i>	
AclManager::\$dialogdef	162
AclManager::\$dialog	162
AclManager::\$dialogdef_areas	162
AclManager::\$dialogdef_areas_total	162
AclManager::\$header	162
AclManager::\$a_params_save	161
AclManager::\$area_view_enabled	161
AclManager::\$acl_id	160
AclManager::\$acl_type	161
AclManager::\$area_view_areas_open	161
AclManager::\$area_view_a_params	161
AclManager::\$intro	163
AclManager::\$output	163
AclManager::calc_areas_total()	165
<i>calculate the total number of items (site, areas, nodes) to show in dialog</i>	
AclManager::build_tree()	165
<i>build a tree of all nodes in an area</i>	
AclManager::dialog_tableform()	166
<i>construct a form with a dialog in a table with 2 or 3 columns</i>	
AclManager::enable_area_view()	167
<i>further initialise the AclManager and enable the area expand/collapse feature</i>	
AclManager::enable_pagination()	167
<i>further initialise the AclManager and enable the dialog pagination feature</i>	
AclManager::\$related_acls	164
AclManager::\$pagination_total	164
AclManager::\$pagination_a_params	163
AclManager::\$pagination_enabled	163
AclManager::\$pagination_limit	163
AclManager::\$pagination_offset	164
admin.php	2
<i>/admin.php - the main entypoint for website maintenance</i>	

B

BY_EMAIL	76
<i>this selects authentication via username+email in authenticate_user()</i>	
BY_LAISSEZ_PASSER	77
<i>this selects authentication via username+laissez_passer in authenticate_user()</i>	
BY_PASSWORD	77
<i>this selects authentication via username+password in authenticate_user()</i>	
build_tree()	116
<i>construct a tree of nodes in memory</i>	
BUTTON_YES	49

BUTTON_SAVE	49
BUTTON_DELETE	49
BUTTON_GO	49
BUTTON_NO	49
BUTTON_OK	49
BUTTON_CANCEL	49

C

constructor ConfigAssistant::ConfigAssistant()	217
<i>constructor for the configuration assistant</i>	
ConfigAssistant::\$where	217
ConfigAssistant::\$table	216
ConfigAssistant::get_dialogdef()	217
<i>construct an array with the dialog information</i>	
ConfigAssistant::get_extra()	217
<i>construct an array based on name=value pairs in an 'extra' field</i>	
ConfigAssistant::show_dialog()	218
<i>add a complete dialog to the content area of the output</i>	
ConfigAssistant::save_data()	218
<i>save the modified configuration parameters to the database</i>	
ConfigAssistant::get_options_from_extra()	218
ConfigAssistant::\$records	216
ConfigAssistant::\$prefix	216
ConfigAssistant::\$dialogdef	215
ConfigAssistant	210
<i>class for editing standard configuration tables</i>	
constructor AreaManager::AreaManager()	201
<i>construct an AreaManager object</i>	
ConfigAssistant::\$dialogdef_hidden	215
ConfigAssistant::\$fields	215
ConfigAssistant::\$language_domain	216
ConfigAssistant::\$keyfield	216
constructor DatabaseMysql::DatabaseMysql()	221
<i>initialise query counter and other variables, store the table prefix</i>	
constructor DatabaseMysqlResult::DatabaseMysqlResult()	229
<i>constructor</i>	
constructor Zip::Zip()	373
<i>constructor initialises all variables</i>	
constructor UserManager::UserManager()	357
<i>construct a UserManager object</i>	
constructor Useraccount::Useraccount()	353
<i>get pertinent user information in core</i>	
constructor InstallWizard::InstallWizard()	395
<i>constructor</i>	
constructor FCKeditor::FCKeditor()	460
<i>Main Constructor.</i>	
<i>Refer to the _samples/php directory for examples.</i>	
CREDITS	495
CHANGES	474
constructor TranslateTool::TranslateTool()	337
<i>construct a TranslateTool object</i>	

constructor Theme::Theme()	325
<i>construct a Theme object</i>	
constructor GroupManager::GroupManager()	265
<i>construct a GroupManager object</i>	
constructor FileManager::FileManager()	246
<i>construct a FileManager object (called from /program/main_admin.php)</i>	
constructor Email::Email()	233
<i>constructor resets all variables to a known (default) state</i>	
constructor Language::Language()	276
<i>constructor</i>	
constructor Module::Module()	282
constructor PageManager::PageManager()	286
<i>construct a PageManager object (called from /program/main_admin.php)</i>	
constructor Node::Node()	283
constructor Area::Area()	193
<i>construct an Area object</i>	
constructor AdminOutput::AdminOutput()	180
<i>constructor</i>	
CAPACITY_CUSTOM7	115
CAPACITY_CUSTOM6	115
CAPACITY_CUSTOM5	115
CAPACITY_CUSTOM8	115
CAPACITY_CUSTOM9	115
CAPACITY_MEMBER	115
CAPACITY_EDITOR	115
CAPACITY_CUSTOM4	115
CAPACITY_CUSTOM3	115
configurationmanagerlib.php	27
<i>/program/lib/configurationmanagerlib.php - configurationmanager</i>	
configassistant.class.php	26
<i>/program/lib/configassistant.class.php - dealing with lists of configuration parameters</i>	
cron.php	7
<i>/cron.php - the main entrypoint for processing cron jobs</i>	
CHORE_SAVE	27
CAPACITY_CHAIR	115
CAPACITY_CUSTOM2	115
CAPACITY_CUSTOM1	115
CAPACITY_NEXT_AVAILABLE	115
CAPACITY_NONE	115
<i>The constants CAPACITY_* are used for group memberships (see accountmanagerlib.php).</i>	
cron_send_queued_alerts()	119
<i>send pending messages/alerts</i>	
convert_to_type()	118
<i>convert a string to another type (bool, int, etc.)</i>	
capacity_name()	118
<i>translate a numeric capacity code to a readable name</i>	
calculate_area()	148
<i>try to retrieve a valid area record based on values of requested area and requested node</i>	
calculate_default_page()	149
<i>try to find a default page within a subtree of pages and sections</i>	
constructor AclManager::AclManager()	164
<i>constructor for the AclManager</i>	
calculate_node_id()	149

<i>calculate and validate the node_id to display</i>	118
calc_user_related_acls()	118
<i>calculate an array with acls related to user \$user_id via group memberships</i>	
calculate_uri_shortcuts()	117
<i>try to eliminate the scheme and authority from the two main uri's</i>	
CAPACITY_PUBLISHER	115
CAPACITY_PROJECTLEAD	115
CAPACITY_PRINCIPAL	115
CAPACITY_PUPIL	115
CAPACITY_SECRETARY	115
CAPACITY_TREASURER	115
CAPACITY_TEACHER	115
config-example.php	3
<i>/config-example.php - example of the main configuration file</i>	

D

DatabaseMysql::close()	222
<i>close the connection to the database</i>	
DatabaseMysql::\$query_counter	221
DatabaseMysql::\$prefix	221
DatabaseMysql::\$error	221
DatabaseMysql::column_definition()	222
<i>convert a fielddef array to a MySQL specific column definition</i>	
DatabaseMysql::concat()	223
<i>helper function for string concatenation in sql statements</i>	
DatabaseMysql::drop_table()	226
<i>unconditionally drop the specified table</i>	
DatabaseMysql::create_table_sql()	225
<i>create the MySQL-specific SQL statement to create a table via a generic table definition</i>	
DatabaseMysql::create_table()	225
<i>create a table via a generic (non-MySQL-specific) table definition</i>	
DatabaseMysql::connect()	224
<i>connect to the database server and open the database</i>	
DatabaseMysql::\$errno	220
DatabaseMysql::\$debug	220
DatabaseMysql	219
<i>MySQL database</i>	
download_source_tree()	143
<i>workhorse function to recursively add most of a tree to a ZIP-archive</i>	
download_source()	142
<i>construct a zipfile with the current source and stream it to the visitor</i>	
DIALOG_NODE_EDIT_CONTENT	97
DatabaseMysql::\$db_link	219
DatabaseMysql::\$db_name	219
DatabaseMysql::\$db_username	220
DatabaseMysql::\$db_type	220
DatabaseMysql::\$db_server	220
DatabaseMysql::\$db_password	219
DatabaseMysql::dump()	226
<i>make a text dump of our tables in the database suitable for backup purposes</i>	
DatabaseMysql::escape()	226

escape special characters in string	
demodata()	382
insert basic demonstration data; the foundation for the module/theme demonstration data	
demodata.php	382
/program/install/demodata.php - code to install the main demodata	
DatabaseMysqlResult::fetch_row_assoc()	230
fetch the next result row as a associative array	
DatabaseMysqlResult::fetch_row()	230
fetch the next result row as a 0-based enumerated array	
demodata_alerts()	383
create a few alerts	
demodata_areas()	383
create three areas + themes	
demodata.php	389
/program/install/languages/nl/demodata.php - translated messages for	
/program/install/demodata.php (Dutch)	
demodata.php	387
/program/install/languages/en/demodata.php - translated messages for	
/program/install/demodata.php (English)	
demodata_users_groups()	384
create a handful of users/groups/capacities/acls	
demodata_sections_pages()	383
create a few sections and pages	
DatabaseMysqlResult::fetch_all_assoc()	230
fetch all rows as an array (0-based or keyed) of associative arrays	
DatabaseMysqlResult::fetch_all()	230
fetch all rows as a 0-based array of 0-based enumerated arrays	
DatabaseMysql::table_exists()	228
see if the named table exists	
DatabaseMysql::query()	228
execute a select query and return a result set	
DatabaseMysql::last_insert_id()	227
retrieve the most recent automatically inserted id ('auto_increment')	
DatabaseMysql::exec()	227
execute an action query and return the number of affected rows	
DatabaseMysqlResult	228
MySQL database result	
DatabaseMysqlResult::\$errno	229
DatabaseMysqlResult::close()	230
free the memory associated with the result set	
DatabaseMysqlResult::\$result	229
DatabaseMysqlResult::\$num_rows	229
DatabaseMysqlResult::\$error	229
DIALOG_NODE_EDIT_ADVANCED	97
DIALOG_NODE_EDIT	97
db_update_sql()	36
generate sql to update one or more fields in a table	
db_update()	36
update one or more fields in a table	
db_select_sql()	35
generate the necessary SQL-code for a simple SELECT statement	
db_select_single_record()	34
fetch a single record from the database	

db_where_clause()	37
construct a where clause from string/array, including the word WHERE	
dbsessionlib.php	39
/program/lib/dbsessionlib.php - functions to keep PHP-sessions in the database	
dbsession_exists()	41
check to see if \$session_key exists in the session table	
dbsession_destroy()	41
remove a session record from the sessions table (it should still exist)	
dbsession_create()	40
create a new session in the session table, return the unique sessionkey	
dbsession_close()	40
'close' a session that was opened with dbsession_open() before	
db_select_all_records()	34
fetch all selected records from the database in one array	
db_last_insert_id()	33
wrapper for DB->last_insert_id()	
db_delete()	30
delete zero or more rows in a table	
db_bool_is()	30
check boolean field in a database-independent way	
database_factory()	29
manufacture a database object	
databaselib.php	29
/program/lib/database/databaselib.php - database factory and database access routines	
db_delete_sql()	31
generate SQL to delete zero or more rows in a table	
db_errormessage()	31
retrieve the latest database error from \$DB	
db_insert_into_sql()	33
generate the necessary SQL-code for an INSERT INTO statement	
db_insert_into_and_get_id()	32
execute the necessary SQL-code for an INSERT INTO statement and return the last_insert_id	
db_insert_into()	32
execute the necessary SQL-code for an INSERT INTO statement	
db_escape_and_quote()	31
conditionally quote and escape values for use with a database table	
dbsession_expire()	41
remove all sessions that were created more than \$max_life seconds ago	
dbsession_garbage_collection()	42
remove all sessions that are last accessed more than \$time_out seconds ago	
dialog_get_widget_richtextinput()	55
construct an input field using the user's preferred editor	
dialog_get_widget_radiocheckbox()	54
construct a checkbox or 1 or more radiobuttons	
dialog_get_widget_listbox()	53
construct a listbox	
dialog_get_widget_file()	52
construct an input field for file upload	
dialog_get_widget_submit()	56
construct a submit button	
dialog_get_widget_textinput()	57
construct an input field, usually for text input OR a textarea for multiline input	
DIALOG_NODE_DELETE_CONFIRM	97

DIALOG_NODE_ADD	97
dialog_validate()	58
<i>validate and check values that were submitted via a user dialog</i>	
dialog_quickform()	58
<i>construct a generic form with a dialog</i>	
dialog_get_widget()	51
<i>construct an actual HTML input widget for dialog input element</i>	
dialog_get_label()	51
<i>construct a label for a dialog input element</i>	
dbsession_remove_obsolete_sessions()	43
<i>workhorse for removing obsolete sessions from the database</i>	
dbsession_read()	43
<i>read the (serialised) session data from the database</i>	
dbsession_open()	43
<i>'open' a session</i>	
dbsession_get_session_id()	42
<i>retrieve the session_id (pkey) that corresponds with session_key</i>	
dbsession_setup()	44
<i>setup database based handlers for session management</i>	
dbsession_write()	44
<i>write the (serialised) data to the database</i>	
dialog_get_class()	51
<i>construct a space-delimited list of classes that apply to this item</i>	
dialog_buttondef()	50
<i>shortcut for generating a dialogdef for a button</i>	
dialoglib.php	46
<i>/program/lib/dialoglib.php - useful functions for manipulating dialogs</i>	
diff_microtime()	10
<i>Calculate the difference between two microtimes</i>	

E

Email::rfc5322_address()	238
<i>construct an address field according to RFC5322 (RFC822)</i>	
Email::rfc5322_message_id()	239
<i>construct a message-id conforming to RFC5322 (RFC2822, RFC822)</i>	
Email::send()	239
<i>send the message using the prepared information (To:, Subject:, the message and attachments etc.)</i>	
Email::rfc2047_qstring()	236
<i>encode a string according to RFC2047 (Message Header Extensions for Non-ASCII Text)</i>	
Email::rfc2047_qchar()	235
<i>encode an 8-bit byte according to Q-encoding in RFC2047</i>	
Email::is_7bit()	234
<i>a small utility routine to determine if a string has only 7bit characters</i>	
Email::reset_all()	235
<i>reset all variables to their default values</i>	
Email::set_header()	241
<i>manually add a header to the mail message</i>	
Email::set_mailfrom()	241
<i>record the address and the name for the From: header</i>	
en_manifest.php	416

<i>/program/languages/en/en_manifest.php - description of the main language/translation (English)</i>	
error_handler()	455
escape_quote()	455
Email::set_subject()	243
<i>store the subject of the mail message</i>	
Email::set_message()	242
<i>set the message</i>	
Email::set_mailreplyto()	242
<i>record the address and the name for the Reply-To: header</i>	
Email::set_mailto()	242
<i>record the address and the name for the To: header</i>	
Email::add_mailcc()	234
<i>add an address and name for the Cc: header</i>	
Email::add_attachment()	233
<i>add an attachment</i>	
Email::\$charset	231
Email::\$eol	231
Email::\$headers	232
Email::\$attachments	231
Email	230
<i>Email implements a simple interface to send mail</i>	
email.class.php	61
<i>/program/lib/email.class.php - wrapper for sending mail</i>	
error_exit404()	143
<i>exit with a 404 not found error</i>	
Email::\$mailcc	232
Email::\$mailfrom	232
Email::\$minimal	233
Email::\$subject	233
Email::\$message	233
Email::\$max_length	233
Email::\$mailreplyto	232
Email::\$mailto	232
error_exit()	10
<i>emergency exit of program in case there is something really, really wrong</i>	

F

FileManager::task_remove_file()	260
<i>show a confirmation dialog for deleting a single file</i>	
FileManager::task_remove_multiple_files()	261
<i>show confirmation dialog for multiple file delete OR perform actual file delete</i>	
FileManager::task_remove_directory()	260
<i>show a confirmation dialog for removing a single directory OR actually removes a directory</i>	
FileManager::task_preview_file()	260
<i>preview a file via file.php</i>	
FileManager::task_list_directory()	260
<i>show a directory listing of the current working directory and links to add/delete files/directories etc.</i>	
FileManager::unique_filename()	261
<i>construct a unique filename taking existing files into account</i>	

FileManager::valid_path()	262
access control and validation for selected directory or file	
frugal_install.php	448
/program/themes/frugal/frugal_install.php -- installer of the frugal theme	
FileManager::vpath()	264
translate a path to the corresponding virtual path	
FileManager::vname()	263
construct the (possibly translated) name of the last directory in the path	
FileManager::virusscan()	262
scan a file for viruses	
FileManager::task_change_directory()	260
make another directory the current (working) directory and optionally change the sort order	
FileManager::task_add_file()	259
add one or more new files to a directory	
FileManager::show_directories()	256
output a simple list of directories (for navigation only)	
FileManager::show_dialog_confirm_delete_files()	255
show a dialog that ask the user to confirm a mass file delete	
FileManager::show_dialog_confirm_delete_directory()	255
show a dialog that ask the user to confirm the removal of a directory	
FileManager::show_breadcrumbs()	255
display a clickable path to the directory \$path	
FileManager::show_directories_and_files()	256
display a list of subdirectories and files in directory \$path	
FileManager::show_file_as_thumbnail()	257
show a thumbnail of a single (image) file perhaps including clickable links for selection in FCK Editor	
FileManager::task_add_directory()	259
create a new subdirectory	
FileManager::sort_entries()	259
sort directory entries	
FileManager::show_menu()	259
show a menu that is equivalent with the root directory	
FileManager::show_list()	258
display a list of directories and files in \$path	
frugal_demodata()	448
add demonstration data to the system	
frugal_install()	449
install the theme	
FCKEditor::\$Width	460
Width of the FCKEditor.	
Examples: 100%, 600	
FCKEditor::\$Value	460
Initial value.	
FCKEditor::\$ToolbarSet	460
Name of the toolbar to load.	
FCKEditor::\$InstanceName	459
Name of the FCKEditor instance.	
FCKEditor::Create()	460
Display FCKEditor.	
FCKEditor::CreateHtml()	460
Return the HTML code required to run FCKEditor.	
FAQ	469

FCKEditor::IsCompatible()	461
<i>Returns true if browser is compatible with FCKEditor.</i>	
FCKEditor::GetConfigFieldString()	461
<i>Get settings from Config array as a single string.</i>	
FCKEditor::EncodeConfig()	460
<i>Encode characters that may break the configuration string generated by GetConfigFieldString().</i>	
FCKEditor::\$Height	459
<i>Height of the FCKEditor.</i>	
<i>Examples: 400, 50%</i>	
FCKEditor::\$Config	459
<i>This is where additional configuration can be passed.</i>	
frugal.php	451
<i>/program/themes/frugal/languages/en/frugal.php - translated messages for theme (English)</i>	
frugal_manifest.php	450
<i>/program/themes/frugal/frugal_manifest.php - description of the frugal theme</i>	
frugal_upgrade()	449
<i>upgrade the theme</i>	
frugal_uninstall()	449
<i>uninstall the theme</i>	
frugal.php	452
<i>/program/themes/frugal/languages/nl/frugal.php - translated messages for theme (Nederlands)</i>	
fckeditor.php	457
FCKEditor::\$BasePath	459
<i>Path to FCKEditor relative to the document root.</i>	
FCKEditor	459
FCKEditor_IsCompatibleBrowser()	458
<i>Check if browser is compatible with FCKEditor.</i>	
<i>Return true if is compatible.</i>	
fckeditor_php4.php	458
FileManager::sanitise_filetype()	254
<i>try to make sure that the extension of file \$name makes sense or matches the actual filetype</i>	
FileManager::make_thumbnail()	253
<i>try to create a thumbnail of the image in file \$filename (best effort)</i>	
FileManager	243
<i>File Manager</i>	
filemanager.class.php	65
<i>/program/lib/filemanager.class.php - filemanager</i>	
filelib.php	62
<i>/program/lib/filelib.php - utilities for manipulating files</i>	
F_TIME	49
FileManager::\$areas	244
FileManager::\$current_directory	244
FileManager::\$output	245
FileManager::\$job	244
FileManager::\$ext_allow_upload	244
FileManager::\$ext_allow_browse	244
F_SUBMIT	49
F_RICHTEXT	49
F_DATETIME	49
F_DATE	49
F_CHECKBOX	49
F_ALPHANUMERIC	49

F FILE	49
F INTEGER	49
F REAL	49
F RADIO	49
F PASSWORD	49
F LISTBOX	49
FileManager::\$show_thumbnails	245
FileManager::\$sort	245
FileManager::get_entries()	251
<i>generate a list of selected files and subdirectories in \$path</i>	
FileManager::get_dialogdef_add_files()	250
<i>construct a dialog definition for adding (uploading) files</i>	
FileManager::file_url()	250
<i>construct a url that links to a file via /file.php</i>	
FileManager::explode_path()	250
<i>shorthand for splitting a path into an array with path components</i>	
FileManager::get_entries_areas()	251
<i>generate a list of (virtual) directories for areas the user can access</i>	
FileManager::get_entries_groups()	252
<i>generate a list of (virtual) directories for groups the user can access</i>	
FileManager::human_readable_size()	253
<i>convert an integer filesize to a human readable form</i>	
FileManager::has_allowed_extension()	253
<i>see if the filename extension is allowed</i>	
FileManager::get_entries_users()	252
<i>generate a list of (virtual) directories for users this user can access</i>	
FileManager::get_entries_root()	252
<i>generate a list of (virtual) directories at the root level</i>	
FileManager::delete_files()	249
<i>workhorse function that actually deletes files, and possibly the corresponding thumbnails</i>	
FileManager::delete_directory()	249
<i>workhorse function that actually removes directories</i>	
FileManager::cmp_entries_bydate_asc()	247
<i>callback for comparing two directory entries by mtime</i>	
FileManager::allowed_extensions()	246
<i>convert a comma-delimited list of allowable extensions to an array (or FALSE if none are allowed)</i>	
FileManager::\$vpaths	245
FileManager::\$usergroups	245
FileManager::cmp_entries_bydate_desc()	247
<i>callback for comparing two directory entries by date (descending)</i>	
FileManager::cmp_entries_byfile_asc()	247
<i>callback for comparing two directory entries by filename</i>	
FileManager::cmp_groups()	248
<i>callback for comparing two group records</i>	
FileManager::cmp_entries_bysize_desc()	248
<i>callback for comparing two directory entries by size (descending)</i>	
FileManager::cmp_entries_bysize_asc()	248
<i>callback for comparing two directory entries by size</i>	
FileManager::cmp_entries_byfile_desc()	247
<i>callback for comparing two directory entries by filename (descending)</i>	
file.php	8
<i>/file.php - the main entrypoint for serving files</i>	

G

GroupManager::get_options_capacities()	270
GroupManager::get_icon_edit()	270
<i>construct a clickable icon to edit the properties of this group</i>	
GroupManager::get_icon_delete()	269
<i>construct a clickable icon to delete this group</i>	
GroupManager::groups_overview()	270
<i>display list of existing groups and an option to add a group</i>	
GroupManager::group_add()	271
<i>present 'add group' dialog where the user can enter minimal properties for a new group</i>	
GroupManager::group_edit()	272
<i>show a dialog with the basic properties of a group</i>	
GroupManager::group_delete()	271
<i>delete a group after confirmation</i>	
GroupManager::get_group_capacity_records()	269
<i>return an array of group-capacity records (possibly buffered)</i>	
GroupManager::get_group_capacity_names()	269
GroupManager::capacity_save()	268
<i>save data from a dialog for a group/capacity</i>	
GroupManager::capacity_pagemanager()	267
<i>show a dialog for modifying page manager permissions for a group/capacity</i>	
GroupManager::delete_group_capacities_acls()	268
<i>actually remove a group and all associated data</i>	
GroupManager::get_dialogdef_add_group()	268
<i>construct the add group dialog</i>	
GroupManager::get_groupname()	269
GroupManager::get_dialogdef_edit_group()	269
<i>construct the edit group dialog</i>	
GroupManager::group_save()	272
<i>save an edited group to the database, including adding/modifying/deleting group/capacity-records</i>	
GroupManager::group_savenew()	272
<i>save a new group to the database</i>	
guestbook_disconnect()	421
<i>disconnect this module from a node</i>	
guestbook_connect()	420
<i>connect this module to a node</i>	
guestbook_admin.php	420
<i>/program/modules/guestbook/guestbook_admin.php - management interface for module</i>	
guestbook_save()	421
<i>save the modified content data of this module linked to node \$node_id</i>	
guestbook_show_edit()	422
<i>present the user with a dialog to modify the content that is connected to node \$node_id</i>	
guestbook.php	425
<i>/program/modules/guestbook/languages/nl/guestbook.php - translated messages for module (Dutch)</i>	
guestbook.php	424
<i>/program/modules/guestbook/languages/en/guestbook.php - translated messages for module (English)</i>	
GroupManager::valid_group_capacity()	275
GroupManager::show_parent_menu()	275
GroupManager::show_breadcrumbs_addgroup()	273

<i>display breadcrumb trail that leads to the add new group dialog</i>	
<u>GroupManager::has_job_permission()</u>	273
<i>determine whether a group/capacity has permissions for a particular job</i>	
<u>GroupManager::show_breadcrumbs_group()</u>	273
<i>display breadcrumb trail that leads to groups overview screen</i>	
<u>GroupManager::show_breadcrumbs_groupcapacity()</u>	274
<i>display breadcrumb trail that leads to group capacity overview screen</i>	
<u>GroupManager::show_menu_groupcapacity()</u>	275
<u>GroupManager::show_menu_group()</u>	274
<i>show a menu for a group including links to the group's capacity overview screens</i>	
<u>GroupManager::capacity_overview()</u>	267
<i>display an overview of all members of a group with a particular capacity</i>	
<u>GroupManager::capacity_intranet()</u>	267
<i>show a dialog for modifying intranet permissions for a group/capacity</i>	
<u>GROUP_SELECT_ALL_USERS</u>	114
<i>this value is used to select all users rather than users from a specific group</i>	
<u>GROUPMANAGER_MAX_CAPACITIES</u>	67
<i>this defines the maximum number of capacities a group can have (keep this below 10 because of dialog hotkeys)</i>	
<u>groupmanager.class.php</u>	67
<i>/program/lib/groupmanager.class.php - taking care of group management</i>	
<u>GROUP_SELECT_NO_GROUP</u>	114
<i>this value is used to select the users that are not associated with any group</i>	
<u>get_area_records()</u>	120
<i>retrieve a list of all available area records keyed by area_id</i>	
<u>get_parameter_string()</u>	120
<i>return an (unquoted) string value specified in the page request or default value if none</i>	
<u>get_parameter_int()</u>	120
<i>return an integer value specified in the page request or default value if none</i>	
<u>get_mimetypes_array()</u>	63
<i>return an array with mimetypes keyed by file extension</i>	
<u>get_mimetype()</u>	62
<i>determine the mimetype of a file</i>	
<u>GROUPMANAGER_DIALOG_CAPACITY_INTRANET</u>	19
<u>GROUPMANAGER_DIALOG_CAPACITY_ADMIN</u>	19
<u>GROUPMANAGER_DIALOG_CAPACITY_PAGEMANAGER</u>	19
<u>GROUPMANAGER_DIALOG_DELETE</u>	19
<u>get_mediatype()</u>	62
<i>extract the mediatype and -subtype from a full mimetype</i>	
<u>GROUPMANAGER_DIALOG_EDIT</u>	19
<u>get_properties()</u>	121
<i>retrieve typed properties (name-value-pairs) from a table</i>	
<u>get_requested_area()</u>	121
<i>get the number of the area the user requested or null if not specified</i>	
<u>GroupManager::add_group_capacity()</u>	265
<u>GroupManager::acl_delete()</u>	265
<i>remove all records relating to 1 or more acl_id's from various acl-tables</i>	
<u>GroupManager::\$show_parent_menu</u>	265
<u>GroupManager::areas_expand_collapse()</u>	266
<i>manipulate the current state if indicator(s) for 'open' and 'closed' areas</i>	
<u>GroupManager::a_params()</u>	266
<i>shorthand for the anchor parameters that lead to the group manager</i>	
<u>GroupManager::capacity_admin()</u>	267

<i>show a dialog for modifying admin permissions for a group/capacity</i>	
GroupManager::calc_acl_id()	266
GroupManager::\$output	264
GroupManager::\$group_capacity_records	264
get_requested_node()	122
<i>get the number of the node the user requested or NULL if not specified</i>	
get_requested_filename()	121
<i>get the name of the requested file</i>	
get_unique_number()	122
<i>a small utility routine that returns a unique integer</i>	
get_user_groups()	122
<i>retrieve the records of the groups of which user \$user_id is a member</i>	
GroupManager	264
<i>Group management</i>	
get_versioncheck_url()	139
<i>construct URL for version check against the project's website</i>	
GROUPMANAGER_DIALOG_ADD	19
<i>Distinguish between the various dialogs</i>	

H

htmlpage_install.php	432
<i>/program/modules/htmlpage/htmlpage_install.php - installer of the htmlpage module</i>	
htmlpage_demodata()	432
<i>add demonstration data to the system</i>	
htmlpage_install()	433
<i>install the module</i>	
htmlpage_uninstall()	433
<i>uninstall the module</i>	
htmlpage_cron()	431
<i>routine that is called periodically by cron</i>	
htmlpage_cron.php	431
<i>/program/modules/htmlpage/htmlpage_cron.php - interface to the cron-part of the htmlpage module</i>	
htmlpage_disconnect()	428
<i>disconnect this module from a node</i>	
htmlpage_save()	428
<i>save the modified content data of this module linked to node \$node_id</i>	
htmlpage_show_edit()	429
<i>present the user with a dialog to modify the content that is connected to node \$node_id</i>	
htmlpage_upgrade()	433
<i>upgrade the module</i>	
htmlpage_manifest.php	434
<i>/program/modules/htmlpage/htmlpage_manifest.php - description of the htmlpage module</i>	
htmlpage.php	438
<i>/program/modules/htmlpage/languages/en/htmlpage.php - translated messages for module (English)</i>	
htmlpage.php	439
<i>/program/modules/htmlpage/languages/nl/htmlpage.php - translated messages for module (Dutch)</i>	
HISTORY	473
htmlpage_tabledefs.php	437

/program/modules/htmlpage/install/htmlpage_tabledefs.php	- data definition for module	
htmlpage_view()	display the content of the htmlpage linked to node \$node_id	436
htmlpage_search.php		435
/program/modules/htmlpage/htmlpage_search.php	- interface to the search-part of the htmlpage module	
htmlpage_search()	search the content of the htmlpage linked to node \$node_id	435
htmlpage_view.php		436
/program/modules/htmlpage/htmlpage_view.php	- interface to the view-part of the htmlpage module	
htmlpage_connect()	connect this module to a node	427
htmlpage_admin.php		427
/program/modules/htmlpage/htmlpage_admin.php	- management interface for htmlpage-module	
html_img()	construct an HTML IMG tag with optional attributes	70
html_input_radio()	STUB	70
html_input_select()	STUB	71
html_input_submit()	STUB	71
html_form_close()	companion of html_form: close the tag	70
html_form()	construct the opening of a HTML form	69
href()	construct a href from a path, params and a fragment	68
html_a()	construct an HTML A tag with optional parameters and attributes	68
html_attributes()	convert an array of name-value pairs to a string	69
html_input_text()	STUB	71
html_table()	construct the opening of a HTML table	71
html_table_row()		72
html_table_row_close()		72
html_tag()	construct a generic HTML-tag with attributes, optionally close it too	72
html_table_head_close()		72
html_table_head()		72
html_table_cell()		72
html_table_cell_close()		72
html_table_close()	construct table closing tag	72
htmlilib.php		68
/program/lib/htmlilib.php	- useful functions for generating HTML-code	

InstallWizard::magic_unquote()	404
<i>this circumvents the 'magic' in magic_quotes_gpc() by conditionally stripping slashes</i>	
InstallWizard::is_already_installed()	404
<i>check for previous install</i>	
InstallWizard::insert_tabledata()	404
<i>fill tables in database via include()'ing a file with tabledata</i>	
InstallWizard::guess_url()	404
<i>educated guesses for scheme, host and portname from \$_SERVER</i>	
InstallWizard::perform_installation()	405
<i>perform the actual initialisation of the cms</i>	
InstallWizard::quasi_random_string()	406
<i>generate a string with quasi-random characters</i>	
InstallWizard::sanitise_filename()	407
<i>sanitise a string to make it acceptable as a filename/directoryname</i>	
InstallWizard::run()	406
<i>main dispatcher for the Installation Wizard</i>	
InstallWizard::render_dialog()	406
<i>quick and dirty dialogdef renderer</i>	
InstallWizard::get_page()	403
<i>construct a complete HTML-page that can be sent to the user's browser</i>	
InstallWizard::get_options_db_type()	403
<i>construct a list of database options</i>	
InstallWizard::get_dialogdef_installtype()	401
<i>fill an array with necessary information for installtype dialog</i>	
InstallWizard::get_dialogdef_finish()	400
<i>fill an array with necessary information for finish / jump dialog</i>	
InstallWizard::get_dialogdef_database()	400
<i>fill an array with necessary information for the database dialog</i>	
InstallWizard::get_dialogdef_language()	401
<i>fill an array with necessary information for language dialog</i>	
InstallWizard::get_dialogdef_user()	401
<i>fill an array with necessary information for the first user dialog</i>	
InstallWizard::get_menu()	402
<i>construct a clickable menu which helps the user to jump back and forth in the funnel</i>	
InstallWizard::get_manifests()	401
<i>retrieve an array of manifests for modules, themes or languages</i>	
InstallWizard::get_list_of_install_languages()	401
<i>retrieve a list of available languages by querying the file system for install.php translation files</i>	
InstallWizard::save_cms()	407
<i>validate and store the CMS-data the user supplied</i>	
InstallWizard::save_database()	407
<i>validate database information</i>	
InstallWizard::show_dialog_user()	412
<i>construct the dialog for the first user account</i>	
InstallWizard::show_dialog_license()	412
<i>construct a full license agreement and an input where the user must enter 'I agree'</i>	
InstallWizard::show_dialog_language()	412
<i>construct the language selection dialog</i>	
InstallWizard::t()	413
<i>retrieve a translated string with optional parameters filled in</i>	
InstallWizard::validate()	413

<i>minimal validation of data input</i>	474
INSTALL	474
InstallWizard::write_config_php()	414
<i>attempt to write the file config.php in the correct location</i>	
InstallWizard::validate_password()	414
<i>validation of password input</i>	
InstallWizard::show_dialog_installtype()	411
<i>construct the installtype + high visibility selection dialog</i>	
InstallWizard::show_dialog_finish()	411
<i>construct the finish screen</i>	
InstallWizard::save_user()	408
<i>validate and store the data for the first user account</i>	
InstallWizard::save_language()	408
<i>store the selected language</i>	
InstallWizard::save_installtype()	408
<i>store the selected install type + high visibility flag</i>	
InstallWizard::show_dialog_cancelled()	408
<i>show the user that the process has been cancelled</i>	
InstallWizard::show_dialog_cms()	409
<i>construct the dialog for essential cms data (title, paths, e-mail address)</i>	
InstallWizard::show_dialog_database()	410
<i>construct the dialog for database (server, host, username, password, etc.)</i>	
InstallWizard::show_dialog_confirm()	410
<i>construct the overview/confirmation dialog</i>	
InstallWizard::show_dialog_compatibility()	410
<i>construct the comptibility overview</i>	
InstallWizard::get_dialogdef_cms()	400
<i>fill an array with necessary information for the cms dialog</i>	
InstallWizard::get_default_install_values()	399
<i>return an array with default configuration values</i>	
INSTALL_DIALOG_FINISH	380
INSTALL_DIALOG_DOWNLOAD	380
INSTALL_DIALOG_DONE	380
INSTALL_DIALOG_DATABASE	380
INSTALL_DIALOG_INSTALLTYPE	380
INSTALL_DIALOG_LANGUAGE	380
index.php	386
<i>/program/install/index.php - redirector for website installation</i>	
INSTALL_DIALOG_USER	381
INSTALL_DIALOG_LICENSE	380
INSTALL_DIALOG_CONFIRM	380
INSTALL_DIALOG_COMPATIBILITY	380
ini_get_int()	123
<i>return an integer (bytecount) value from PHP ini</i>	
initialise()	11
<i>initialise the program, setup database, read configuration, etc.</i>	
init.php	10
<i>/program/init.php - setup database connection, sessions, configuration, etc.</i>	
is_expired()	123
<i>determine if any of the ancestors or \$node_id itself is already expired</i>	
is_under_embargo()	124
<i>determine if any of the ancestors or \$node_id itself is under embargo</i>	
INSTALL_DIALOG_CMS	380

INSTALL DIALOG CANCELLED	380
install.php	380
<i>/program/install.php - the main entrypoint for website installation</i>	
install.php	388
<i>/program/install/languages/en/install.php - translated messages for /program/install.php (English)</i>	
install.php	390
<i>/program/install/languages/nl/install.php - translated messages for /program/install.php (Dutch)</i>	
InstallWizard::create_tables()	398
<i>create tables in database via include()'ing a file with tabledefs</i>	
InstallWizard::construct_config_php()	398
<i>prepare a configuration file based on the collected information</i>	
InstallWizard::clamscan_installed()	397
<i>try to locate clamdscan or clamscan on the server</i>	
InstallWizard::end_session_and_redirect()	398
<i>unset installation data, end session and redirect the user to elsewhere</i>	
InstallWizard::errorcount_bump()	399
<i>increment the error counter and perhaps slow things down</i>	
InstallWizard::gd_supported()	399
<i>retrieve information about GD and supported graphics file formats</i>	
InstallWizard::fetch_license()	399
<i>helper to retrieve the text of the LICENSE AGREEMENT for Website</i>	
InstallWizard::errorcount_reset()	399
<i>reset the error counter</i>	
InstallWizard::check_validation()	397
<i>shorthand to check the validation status of the relevant dialogs</i>	
InstallWizard::check_license()	397
<i>check if the user accepts the licences</i>	
InstallWizard::\$messages	395
InstallWizard::\$license	395
InstallWizard	394
<i>class for performing installation tasks</i>	
InstallWizard::\$results	395
InstallWizard::appropriate_legal_notices()	396
<i>construct a link to appropriate legal notices as per AGPLv3 section 5</i>	
InstallWizard::check_for_nameclash()	397
<i>check for name clash of new user (webmaster) and user accounts from demodata</i>	
InstallWizard::check_compatibility()	396
<i>check certain compatibility issues and optionally return test results</i>	
InstallWizard::button()	396
<i>shorthand for creating a submit button in the correct style</i>	
index.php	9
<i>/index.php - the main entrypoint for website visitors (frontpage)</i>	

J

JOB_CONFIGURATIONMANAGER	136
<i>This is used to dispatch the configuration manager</i>	
JOB_FILEBROWSER	136
<i>This is used to dispatch the file manager in file browser mode (used with FCK Editor)</i>	
JOB_FILEMANAGER	136
<i>This is used to dispatch the file manager</i>	

<u>JOB_ACCOUNTMANAGER</u>	136
<i>This is used to dispatch the account manager (users and groups)</i>	
<u>javascript_alert()</u>	124
<i>message a message and generate a javascript alert()</i>	
<u>JOB_PERMISSION_TRANSLATETOOL</u>	113
<i>This allows the user to translate the program, by modifying existing translations or adding new languages</i>	
<u>JOB_PERMISSION_UPDATE</u>	113
<i>This allows the user to perform a system upgrade (see also <u>was_version_check()</u> and <u>main_admin()</u>)</i>	
<u>JOB_FLASHBROWSER</u>	136
<i>This is used to dispatch the file manager in flash browser mode (used with FCK Editor)</i>	
<u>JOB_IMAGEBROWSER</u>	136
<i>This is used to dispatch the file manager in image browser mode (used with FCK Editor)</i>	
<u>JOB_TOOLS</u>	137
<i>This is used to dispatch the tool manager</i>	
<u>JOB_UPDATE</u>	137
<i>This is used to dispatch the update manager</i>	
<u>job_start()</u>	139
<i>generate the start centre page</i>	
<u>JOB_STATISTICS</u>	137
<i>This is used to dispatch the statistics</i>	
<u>JOB_STARTCENTER</u>	137
<i>This is used to dispatch the startcenter job</i>	
<u>JOB_MODULEMANAGER</u>	136
<i>This is used to dispatch the module manager</i>	
<u>JOB_PAGEMANAGER</u>	137
<i>This is used to dispatch the page manager</i>	
<u>JOB_PERMISSION_TOOLS</u>	112
<i>combine the permsnsions for the tools in a single bit mask for convenient testing</i>	
<u>JOB_PERMISSION_STATISTICS</u>	112
<i>This permissions allows the user to access the site statistics</i>	
<u>job_update()</u>	108
<i>main entry point for update wizard (called from /program/main_admin.php)</i>	
<u>JOB_PERMISSION_ACCOUNTMANAGER</u>	111
<i>This (dangerous) permission allows access to add/edit/delete users and groups (including escalate privileges)</i>	
<u>JOB_PERMISSION_BACKUPTOOL</u>	111
<i>This allows the user to download a backup of the database</i>	
<u>job_tools()</u>	103
<i>main entry point for tools (called from /program/main_admin.php)</i>	
<u>job_statistics()</u>	99
<i>main entry point for statistics (called from admin.php)</i>	
<u>job_configurationmanager()</u>	27
<i>main entry point for configurationmanager (called from /program/main_admin.php)</i>	
<u>job_modulemanager()</u>	96
<i>main entry point for modulemanager (called from admin.php)</i>	
<u>JOB_PERMISSION_CONFIGURATIONMANAGER</u>	112
<i>This permission allows the user to access the configuration manager and change the site configuration</i>	
<u>JOB_PERMISSION_FILEMANAGER</u>	112
<i>This permission allows the user to access the file manager and upload/delete files in selected places</i>	

JOB_PERMISSION_NEXT_AVAILABLE_VALUE	112
<i>NOTE: This quasi-permission should always be defined to be the highest permission << 1</i>	
JOB_PERMISSION_PAGEMANAGER	112
<i>This permission allows the user to access the page manager and add/edit/delete nodes according to the user's ACLs</i>	
JOB_PERMISSION_STARTCENTER	112
<i>This permission is required for every user that is to logon to admin.php</i>	
JOB_PERMISSION_MODULEMANAGER	112
<i>This permission allows the user to access the module manager and configure modules</i>	
JOB_PERMISSION_MASK	112
<i>This mask can be used to isolate only the 'official' permissions from an integer value</i>	
JOB_PERMISSION_GURU	112
<i>Guru permissions = all permission bits are set, even the unused ones</i>	
JOB_PERMISSION_LOGVIEW	112
<i>This allows the user to view the contents of the log table</i>	
job_accountmanager()	20
<i>main entry point for accountmanager (called from admin.php)</i>	

L

lock_release()	127
<i>unlock a record that was previously successfully locked</i>	
lock_record_node()	126
<i>get record lock on a node</i>	
lock_release_node()	127
<i>release lock on a node</i>	
logger()	127
<i>a simple function to log information to the database 'for future reference'</i>	
Language	275
<i>Translations of messages in different languages</i>	
lock_record()	125
<i>put a (co-operative) lock on a record</i>	
login_stylesheetsheet()	87
<i>a simple in-line style sheet conveniently grouped in a single routine</i>	
login_page_open()	85
<i>construct the start of a simple HTML-page and open a full size table</i>	
login_send_bypass()	85
<i>send a new (temporary) password to the user via email</i>	
login_send_confirmation()	86
<i>send email to user confirming password change</i>	
login_send_laissez_passer()	87
<i>send a special one-time login code to the user via email</i>	
Language::\$default_domain	275
Language::\$languages	276
Language::get_phrases_from_file()	280
<i>return the \$string array after including a file</i>	
Language::get_phrases_from_database()	280
<i>retrieve phrases from the database for specified domain and language</i>	
Language::reset_cache()	281
<i>remove selected entries (per language+domain, per language, or all) from cache</i>	
Language::retrieve_languages()	281
<i>retrieve an array with all active languages from the database</i>	

LICENSE	473
Language::get_phrase()	278
<i>translation of phrases via a phrase key</i>	
Language::get_languages_to_try()	278
<i>calculate a list of possible languages and parent-languages to try for translations</i>	
Language::\$phrases	276
Language::get_active_language_names()	276
<i>return an array with active languages and language names</i>	
Language::get_current_language()	276
<i>determine the default language to use for translation of phrases</i>	
Language::get_filenames_to_try()	277
<i>calculate an ordered list of filenames to try for translation of phrases</i>	
login_page_close()	85
<i>construct the end of the simple HTML-page, closing the full size table</i>	
login_is_blacklisted()	84
<i>find out if a remote address is blacklisted at this time</i>	
LOGIN PROCEDURE MESSAGE BOX	77
<i>this is a pseudo procedure, used to deliver some message to the user</i>	
LOGIN PROCEDURE CHANGE PASSWORD	77
<i>this is the procedure to change the user's password</i>	
LOGIN PROCEDURE NORMAL	77
<i>this is the usual procedure for logging in</i>	
LOGIN PROCEDURE SEND BYPASS	77
<i>this is phase 2 of the 'forgot password' procedure</i>	
LOGIN PROCEDURE SEND LAISSEZ PASSER	77
<i>this is phase 1 of the 'forgot password' procedure</i>	
LOGIN PROCEDURE BLACKLIST	77
<i>this is a pseudo procedure, used to record blacklisted IP-addresses</i>	
LOGIN FAILURE DELAY SECONDS	77
<i>this is the number of seconds to delay responding after a login action fails (slow 'm down..)</i>	
loginlib.php	17
<i>/program/languages/nl/loginlib.php - translated messages for login procedure and change password</i>	
language.class.php	74
<i>/program/lib/language.class.php - taking care of translations of messages</i>	
loginlib.php	75
<i>/program/lib/loginlib.php -- functions to handle user login/logout</i>	
LOGIN DEBUG	77
<i>useful when debugging routines in this file: 0=production, 1=debugging</i>	
LOGIN PROCEDURE SHOWLOGIN	77
<i>this only shows the login dialog</i>	
login_change_password()	80
<i>update the users database with a new (randomly salted) password and reset bypass mode to normal</i>	
login_failure_blacklist_address()	83
<i>add remote_addr to the blacklist for specified interval (in seconds)</i>	
login_dialog_text_input()	82
<i>add a row with an ordinary input field to the login dialog/table</i>	
login_failure_delay()	83
<i>delay execution of this script for a few seconds and blacklist the remote_addr during the delay</i>	
login_failure_increment()	83
<i>add 1 point to score for a particular IP-address and failed procedure, return the new score</i>	
login_failure_reset()	84

<i>deactivate all login failures/blacklisting scores for remote_addr</i>	
login_dialog_submit_input()	82
<i>add a row with a submit button to the login dialog/table</i>	
login_dialog_password_input()	82
<i>add a row with a password input field to the login dialog/table</i>	
login_dialog_close()	80
<i>close the login dialog/table and maybe an opened HTML-form</i>	
login_dialog_home_forgot_password()	80
<i>add a row with links to home page and forgot password dialog to the login dialog/table</i>	
login_dialog_instruction()	81
<i>add a row to the table/dialog with wordwrap()'ed instruction for the user</i>	
login_dialog_open()	81
<i>construct the start of the login dialog, opening the form and the secondary table</i>	
loginlib.php	14
<i>/program/languages/en/loginlib.php - translated messages for login procedure and change password</i>	

M

Module::\$node_id	282
Module::get_breadcrumb_anchors()	282
<i>get additional breadcrumb trail</i>	
Module::get_content()	282
<i>get the actual content for node \$node_id</i>	
Module::\$module_record	282
Module	281
<i>Methods to access properties of a module</i>	
module_load_view()	150
<i>load the visitor/view interface of a module in core</i>	
module_view()	151
<i>call the routine that generates the view (content) of module \$module_id</i>	
manual.php	153
<i>/program/manual.php - a kickstarter for the documentation</i>	
ModuleHtmlpage	440
<i>/program/modules/htmlpage/htmlpage.class.php - module for plain HTML-pages</i>	
ModuleHtmlpage::get_content()	440
<i>get the actual content for node \$node_id</i>	
mypage_disconnect()	444
<i>disconnect this module from a node</i>	
mypage_save()	445
<i>save the modified content data of this module linked to node \$node_id</i>	
mypage_show_edit()	446
<i>present the user with a dialog to modify the content that is connected to node \$node_id</i>	
mypage_connect()	444
<i>connect this module to a node</i>	
mypage_admin.php	444
<i>/program/modules/mypage/mypage_admin.php - management interface for module</i>	
mypage.php	442
<i>/program/modules/mypage/languages/en/mypage.php - translated messages for module (English)</i>	
mypage.php	443
<i>/program/modules/mypage/languages/nl/mypage.php - translated messages for module</i>	

(Dutch)	
main_index()	150
<i>main program for visitors</i>	
main_index.php	148
<i>/program/main_index.php - workhorse for visitor interface</i>	
module.class.php	93
<i>/program/lib/module.class.php - taking care of modules</i>	
modulelib.php	94
<i>/program/lib/modulelib.php - module factory</i>	
module_factory()	94
<i>manufacture a module object</i>	
MINIMUM_PASSWORD_UPPERCASE	78
<i>this is the hardcoded minimal number of upper case characters in a new password</i>	
MINIMUM_PASSWORD_LOWERCASE	78
<i>this is the hardcoded minimal number of lower case characters in a new password</i>	
MAXIMUM_LINE_LENGTH	77
<i>this defines the maximum line length in messages and instructions</i>	
MINIMUM_PASSWORD_DIGITS	77
<i>this is the hardcoded minimal number of digits in a new password</i>	
MINIMUM_PASSWORD_LENGTH	78
<i>this hardcoded minimal length is enforced whenever a user wants to change her password</i>	
modulemanagerlib.php	96
<i>/program/lib/modulemanagerlib.php - modulemanager</i>	
MODULE_NAME_DEFAULT	97
<i>Default initial module of a new page (see get_dialogdef_add_node())</i>	
main_cron()	141
main_file.php	142
<i>/program/main_file.php - workhorse for serving files</i>	
main_file()	143
<i>main program for serving files</i>	
main_cron.php	141
<i>/program/main_cron.php - take care of recurring jobs</i>	
main_admin()	140
<i>main program for site maintenance</i>	
magic_unquote()	128
<i>this circumvents the 'magic' in magic_quotes_gpc() by conditionally stripping slashes</i>	
main_admin.php	136
<i>/program/main_admin.php - workhorse for site maintenance</i>	
mysql.class.php	38
<i>/program/lib/database/mysql.class.php - access to mysql via database class</i>	

N

Node::exists()	284
Node::calculate_node()	283
<i>determine which node should be displayed based on user's request</i>	
Node::get_area_id()	284
Node::get_node_path()	284
nl_manifest.php	418
<i>/program/languages/nl/nl_manifest.php - description of the Dutch translation package</i>	
Node::\$node_path	283
Node::\$node_exists	283

NODE VISIBILIY EMBARGO	97
<i>Initial visibility of a new node: under embargo</i>	
NODE VISIBILIY HIDDEN	97
<i>Initial visibility of a new node: hidden</i>	
NODE VISIBILIY VISIBLE	97
<i>Initial visibility of a new node: visible</i>	
Node	282
<i>/program/lib/node.class.php - taking care of nodes</i>	
NODE VISIBILIY DEFAULT	97
<i>Default initial visibility of a new node (see <code>get_dialogdef_add_node()</code>)</i>	

P

PageManager::permission_delete_node()	304
<i>does the user have the privilege to delete a node from the area?</i>	
PageManager::permission_edit_node()	304
<i>does the user have the privilege to edit node properties?</i>	
PageManager::permission_add_node()	303
<i>does the user have the privilege to add a node the area or a section?</i>	
PageManager::permission_add_any_node()	303
<i>does the user have the privilege to add a node, any node to an area?</i>	
PageManager::node_has_grandchildren()	303
<i>shorthand to determine whether the number of levels below section <code>\$node_id</code> is greater than one</i>	
PageManager::permission_edit_node_content()	305
<i>does the user have the privilege to edit node content?</i>	
PageManager::permission_set_default()	305
<i>does the user have the privilege to make node <code>\$node_id</code> the default?</i>	
PageManager::section_is_open()	310
<i>shorthand for determing whether a section is opened or closed</i>	
PageManager::save_node_new_area_mass_move()	308
<i>workhorse routine for moving a complete subtree to another area</i>	
PageManager::save_node()	307
<i>workhorse routing for saving modified node data to the database</i>	
PageManager::queue_area_node_alert()	306
<i>add a message to message queue of 0 or more alerts</i>	
PageManager::node_full_name()	302
<i>shorthand for constructing a readable page/section name with id, name and title</i>	
PageManager::module_show_edit()	302
<i>show a dialog for editing the content of module <code>\$module_id</code> linked to page <code>\$node_id</code></i>	
PageManager::get_options_parents_walk()	297
<i>workhorse for construction an options list of possible parent sections</i>	
PageManager::get_options_sort_order()	297
<i>generate a list of siblings in a particular (sub)section used to select/change sort order via a list box</i>	
PageManager::get_options_parents()	296
<i>construct an options list of possible parent sections</i>	
PageManager::get_options_modules()	296
<i>fetch a list of available modules for inclusion on a page</i>	
PageManager::get_options_area()	295
<i>generate a list of areas for use in a dropdown list (for moving a node to another area)</i>	
PageManager::lock_records_subtree()	298

<i>attempt to lock all node records in a subtree</i>	299
PageManager::message_from_lockinfo()	299
<i>construct a readable message from the lockinfo array</i>	
PageManager::module_save()	301
<i>(maybe) save the modified content of module \$module_id connected to page \$node_id</i>	
PageManager::module_load_admin()	301
<i>load the admin interface of a module in core</i>	
PageManager::module_disconnect()	300
<i>inform module \$module_id that it is no longer linked to page \$node_id</i>	
PageManager::module_connect()	299
<i>inform module \$module_id that from now on it will be linked to page \$node_id</i>	
PageManager::show_area_menu()	310
<i>construct a clickable list of available areas for the current user</i>	
PageManager::show_dialog_delete_node_confirm()	310
<i>display a list of 1 or more nodes to delete and ask user for confirmation of delete</i>	
PageManager::task_treeview()	320
<i>maybe change the current area and then show the tree and the menu for the current area</i>	
PageManager::task_treeview_set()	320
<i>this sets the tree view to the specified mode</i>	
PageManager::task_subtree_expand()	320
<i>open the selected section and perhaps change the view mode</i>	
PageManager::task_subtree_collapse()	319
<i>close the selected section and perhaps change the view mode</i>	
PageManager::task_set_default()	319
<i>make the selected node the default for this level</i>	
PROJECT SITE	381
print_checker_results()	455
print_words_elem()	456
print_textinputs_var()	456
print_textindex_decl()	455
print_suggs_elem()	455
PageManager::task_save_node()	319
PageManager::task_save_newnode()	317
<i>save a newly added node to the database</i>	
PageManager::show_tree_walk()	313
<i>display the specified node, optionally all subtrees, and subsequently all siblings</i>	
PageManager::show_treeview_buttons()	312
<i>show one or two clickable links to change the view of the tree</i>	
PageManager::show_tree()	311
<i>create a tree-like list of nodes in the content area of \$this->output</i>	
PageManager::show_edit_menu()	311
<i>construct a clickable list of edit variants (basic, advanced and maybe content)</i>	
PageManager::task_node_add()	314
<i>display a dialog to add a new page or section to the current area</i>	
PageManager::task_node_delete()	314
<i>delete one or more nodes from an area after user confirmation</i>	
PageManager::task_save_content()	317
PageManager::task_page_preview()	316
<i>preview a page that is maybe still under embargo/already expired</i>	
PageManager::task_node_edit_content()	316
<i>display a dialog where the user can edit the contents of a node via a module</i>	
PageManager::task_node_edit()	315
<i>display a dialog where the user can edit basic or advanced properties of a node</i>	

PageManager::get_node_id_and_ancestors()	295
<i>get an array with all ids of ancestors of node_id and node_id itself</i>	
PageManager::get_module_records()	295
<i>retrieve a list of all available module records</i>	
PERMISSION_NODE_ADD_SECTION	113
PERMISSION_NODE_DROP_CONTENT	113
PERMISSION_NODE_ADD_PAGE	113
PERMISSION_NODE_ADD_CONTENT	113
PERMISSION_AREA_EDIT_AREA	113
PERMISSION_NODE_DROP_PAGE	113
PERMISSION_NODE_DROP_SECTION	113
PERMISSION_SITE_ADD_AREA	113
PERMISSION_NODE_EDIT_SECTION	113
PERMISSION_NODE_EDIT_PAGE	113
PERMISSION_NODE_EDIT_CONTENT	113
PERMISSION_AREA_DROP_SECTION	113
PERMISSION_AREA_DROP_PAGE	113
PARAM_SORT	65
password_hash()	87
<i>calculate a hash from a salt and a password</i>	
PARAM_PATH	65
PARAM_FILENAMES	65
<i>This constant is used to construct the fieldname counting the number of files to delete</i>	
PARAM_FILENAME	65
<i>This constant is used to construct the fieldname used for deleting files</i>	
password_hash_check()	88
<i>check equivalency of salt+password against hash</i>	
password_salt()	89
<i>generate a quasi random string to salt the password hash</i>	
PERMISSION_AREA_ADD_SECTION	113
PERMISSION_AREA_ADD_PAGE	113
PARAM_TREEVIEW	97
pagemanager.class.php	97
<i>/program/lib/pagemanager.class.php - pagemanager</i>	
PERMISSION_SITE_DROP_AREA	113
PERMISSION_SITE_EDIT_SITE	113
PageManager::get_dialog_data_node()	291
<i>fill the node dialog with data from the database</i>	
PageManager::get_icon_delete()	291
<i>construct a clickable icon to delete this node (and underlying nodes too)</i>	
PageManager::get_dialogdef_edit_node()	290
<i>construct a dialog definition for editing basic properties of an existing node (page or section)</i>	
PageManager::get_dialogdef_edit_advanced_node()	290
<i>construct a dialog definition for editing advanced properties of a node (page or section)</i>	
PageManager::get_dialogdef_add_node()	289
<i>construct a dialog definition for adding a node (page or section)</i>	
PageManager::get_icon_edit()	292
<i>construct a clickable icon to edit this node</i>	
PageManager::get_icon_home()	292
<i>construct a clickable icon to set the home page/section on this tree level</i>	
PageManager::get_link_node_edit()	294
<i>construct a clickable link to edit this node showing the page's title or link-text</i>	
PageManager::get_icon_section()	294

construct a clickable icon to open/close this node	
PageManager::get_icon_page_preview()	293
construct a clickable icon to preview this node	
PageManager::get_icon_invisibility()	293
construct a clickable icon to edit the advanced properties of this node	
PageManager::delete_node()	289
workhorse routine for deleting a node, including children	
PageManager::calc_home_id()	289
calculate the current default node on this level	
PageManager::\$areas	285
PageManager	285
Page Manager	
performance_get_seconds()	129
return the script execution time	
performance_get_queries()	129
return the number of database queries that was executed	
PageManager::\$area_id	285
PageManager::\$output	285
PageManager::calculate_updated_sort_order()	287
calculate an updated sort order and also make space in the section for moving the node around	
PageManager::calculate_new_sort_order()	286
calculate a new sort order and at the same time make room for a node	
PageManager::build_cached_tree()	286
construct \$this->tree for future reference	
PageManager::\$tree	286
process_task_site()	27
handle the editing/saving of the main configuration information	

Q

quasi_random_string()	129
generate a string with quasi-random characters	
quoted_printable()	130
convert string \$s from native format to quoted printable (RFC2045)	
QUASI_RANDOM_HEXDIGITS	115
QUASI_RANDOM_DIGITS_UPPER_LOWER	115
QUASI_RANDOM_DIGITS_UPPER	115
QUASI_RANDOM_DIGITS	115

R

README	473
rfc1123date()	145
generate an RFC1123-compliant date/time stamp	
readfile_chunked()	145
send a file to the visitor's browser in chunks	
replace_crlf()	132
unfold a possible multiline string	
redirect_and_exit()	131
redirect to another url by sending an http header	

S

show_tools_intro()	103
<i>display an introductory text for tools + menu</i>	
statisticslib.php	99
<i>/program/lib/statisticslib.php - statistics</i>	
show_login()	89
<i>show complete login dialog and exit</i>	
SORTBY_SIZE_DESC	65
show_tools_menu()	104
<i>display the tools menu</i>	
show_update_overview()	108
<i>display an introductory text for update + status overview</i>	
spellchecker.php	455
send_file_from_datadir()	145
<i>the designated file is sent to the visitor</i>	
string2time()	133
<i>convert a string representation of a date/time to a timestamp</i>	
sanitise_filename()	132
<i>sanitise a string to make it acceptable as a filename/directoryname</i>	
SORTBY_SIZE_ASC	65
SORTBY_NONE	65
SQL_FALSE	38
<i>this circumvents the sub-optimal implementation of booleans in MySQL</i>	
show_configuration_menu()	28
<i>display the configuration manager menu</i>	
show_configuration_intro()	28
<i>display an introductory text for the configuration manager + menu</i>	
show_accounts_menu()	20
<i>display the account manager menu</i>	
SQL_TRUE	38
<i>this circumvents the sub-optimal implementation of booleans in MySQL</i>	
SORTBY_DATE_ASC	65
SORTBY_FILE_DESC	65
SORTBY_FILE_ASC	65
SORTBY_DATE_DESC	65
show_accounts_intro()	20
<i>display an introductory text for the account manager + menu</i>	

T

Theme::add_stylesheet()	327
<i>add a link to a stylesheet to the HTML head part of the document</i>	
Theme::calc_breadcrumb_trail()	327
<i>set breadcrumbs in tree AND construct list of clickable anchors</i>	
Theme::calc_tree_visibility()	328
<i>calculate the visibility of the nodes in the tree</i>	
Theme::construct_tree()	328
Theme::add_popup_top()	327
<i>add a message to the list of popup-messages at the TOP of the document</i>	
Theme::add_popup_bottom()	327
<i>add a message to the list of popup-messages at the BOTTOM of the document</i>	

Theme::add_message()	326
<i>add a message to the list of inline messages, part of the BODY of the document</i>	
Theme::add_meta()	326
<i>add a line with meta-information to the HTML head part of the document</i>	
Theme::add_meta_http_equiv()	327
<i>add a line with http-equiv meta-information to the HTML head part of the document</i>	
Theme::dump_subtree()	328
Theme::friendly_bookmark()	329
<i>construct an alphanumeric string from a node title (for a readable bookmark)</i>	
Theme::get_html()	331
<i>construct an output page in HTML</i>	
Theme::get_html_head()	331
<i>get all lines in the HTML head section in a single properly indented string</i>	
Theme::get_jumpmenu()	331
<i>construct a simple jumplist to navigate to other areas</i>	
Theme::get_lines()	332
<i>get lines from an array in a single properly indented string</i>	
Theme::get_div_messages()	330
<i>get a perhaps bulleted list of messages in a DIV</i>	
Theme::get_div_breadcrumbs()	330
<i>construct breadcrumb trail</i>	
Theme::get_address()	329
<i>return the reconstructed URL in a single (indented) line</i>	
Theme::get_bottomline()	329
<i>show 'powered by' and (maybe) report basic performance indicators</i>	
Theme::get_content()	330
<i>get all lines in the content DIV in a single properly indented string</i>	
Theme::add_http_header()	326
<i>add an HTTP-header</i>	
Theme::add_html_header()	326
<i>add a header to the HTML head part of the document</i>	
Theme::\$friendly_url	322
Theme::\$high_visibility	323
Theme::\$html_head	323
Theme::\$http_headers	323
Theme::\$dtd	322
Theme::\$domain	322
Theme::\$breadcrumb_addendum	321
Theme::\$config	321
Theme::\$content	322
Theme::\$messages_bottom	323
Theme::\$messages_inline	323
Theme::\$theme_record	325
Theme::\$title	325
Theme::\$tree	325
Theme::add_content()	326
<i>add a line or array of lines to the content part of the document</i>	
Theme::\$theme_id	324
Theme::\$preview_mode	324
Theme::\$messages_top	324
Theme::\$node_id	324
Theme::\$node_record	324
Theme::get_logo()	332

construct an image tag with the area logo	
Theme::get_menu()	332
TranslateTool::language_add()	343
present the language dialog where the user can enter properties for a new language	
TranslateTool::language_edit()	344
show the language edit dialog	
TranslateTool::language_save()	344
validate and save modified data to database	
TranslateTool::language_savenew()	345
save the newly added language to the database	
TranslateTool::languages_overview()	343
display list of languages with edit icons and an option to add a language	
TranslateTool::guess_row_count()	342
try to calculate a reasonable number of textarea rows based on the contents of \$text	
TranslateTool::get_icon_edit()	341
construct a clickable icon to edit the properties of this language	
TranslateTool::get_options_languages()	341
fetch a list of languages available as parent language	
TranslateTool::get_strings_system()	342
retrieve strings (translations) and comments from an official (system) translation file	
TranslateTool::put_strings_userfile()	345
save new or changed translations to a file under CFG->datadir/languages	
TranslateTool::render_translation_dialog()	345
render a translation dialog based on a dialog definition	
tabledata.php	391
/program/install/tabledata.php defines core data in a generic way	
tabledefs.php	392
/program/install/tabledefs.php defines all core tables in a generic way	
ThemeFrugal	453
/program/themes/frugal/frugal.class.php - the class that comprises the most minimal theme	
TODO	469
TranslateTool::translation_save()	347
save the modified translations in a file in the tree CFG->datadir/languages/	
TranslateTool::translation_edit()	347
show an edit dialog with phrases from \$full_domain in \$language_key	
TranslateTool::show_domain_menu()	346
display the domain menu via \$this->output	
TranslateTool::show_parent_menu()	346
allow the caller to use the menu area (or not)	
TranslateTool::submit_diff_to_project()	346
send new or changed translations back to the project	
TranslateTool::get_domains()	341
return an ordered list of translation domains	
TranslateTool::get_dialogdef_language_domain()	340
construct the translation dialog for selected language and domain	
Theme::get_quicktop()	335
construct a list of quicklinks for top of page (if any)	
Theme::node2anchor()	335
construct an anchor from a node record	
Theme::send_headers()	335
send collected HTTP-headers to user's browser	
Theme::send_output()	336
send collected output to user's browser	

Theme::get_quicklinks()	334
<i>workhorse for constructing list of quicklinks</i>	
Theme::get_quickbottom()	333
<i>construct a list of quicklinks for top of page (if any)</i>	
Theme::get_navigation()	333
Theme::get_popups()	333
<i>construct javascript alerts for messages</i>	
Theme::get_properties()	333
<i>retrieve configuration parameters for this combination of theme and area</i>	
Theme::set_preview_mode()	336
Theme::show_tree_walk()	336
TranslateTool::a_param()	338
<i>shorthand for the anchor parameters that lead to the translate tool</i>	
TranslateTool::code_highlight()	338
<i>highlight code constructs in texts that are to be translated</i>	
TranslateTool::diff_to_text()	339
<i>convert an array with key-value-pairs to a php source file that can be included as a user translation</i>	
TranslateTool::get_dialogdef_language()	340
<i>construct the language dialog (used for both add and edit)</i>	
TranslateTool::\$show_parent_menu	337
TranslateTool::\$output	337
TranslateTool	336
<i>Methods to access properties of a language and modify translations</i>	
TranslateTool::\$domains	337
TranslateTool::\$languages	337
Theme::\$area_record	321
Theme::\$area_id	321
TASK_USER_TREEVIEW	20
TASK_ALERTS	27
TASK_AREAS	27
TASK_CONFIGURATION_INTRO	27
TASK_USER_SAVE_NEW	20
TASK_USER_SAVE	20
TASK_USER_INTRANET	20
TASK_USER_MODULE	20
TASK_USER_PAGEMANAGER	20
TASK_SITE	27
TASK_ADD_DIRECTORY	65
TASK_REMOVE_FILE	65
TASK_REMOVE_MULTIPLE_FILES	65
THUMBNAIL_PREFIX	65
<i>This constant is used to specify thumbnail files to be ignored in directory listings</i>	
TASK_ADD_PAGE	97
TASK_REMOVE_DIRECTORY	65
TASK_PREVIEW_FILE	65
TASK_ADD_FILE	65
TASK_CHANGE_DIRECTORY	65
TASK_LIST_DIRECTORY	65
TASK_USER_GROUPS_SAVE	20
TASK_USER_GROUPS	20
TASK_GROUP_CAPACITY_OVERVIEW	19
<i>TASK_GROUP_CAPACITY_* relate to group-capacity-combinations</i>	

TASK_GROUP_CAPACITY_PAGEMANAGER	19
TASK_GROUP_CAPACITY_SAVE	19
TASK_GROUP_DELETE	19
TASK_GROUP_CAPACITY_MODULE	19
TASK_GROUP_CAPACITY_INTRANET	19
TASK_GROUPS	19
<i>TASK_GROUP* relate to plain groups</i>	
TASK_GROUP_ADD	19
TASK_GROUP_CAPACITY_ADMIN	19
TASK_GROUP_EDIT	19
TASK_GROUP_SAVE	19
TASK_USER_DELETE	20
TASK_USER_EDIT	20
TASK_USER_GROUPADD	20
TASK_USER_GROUPDELETE	20
TASK_USER_ADVANCED	20
TASK_USER_ADMIN	20
TASK_GROUP_SAVE_NEW	19
TASK_USERS	19
<i>TASK_USER* relate to user accounts</i>	
TASK_USER_ADD	20
TASK_ADD_SECTION	97
TASK_NODE_DELETE	97
TRANSLATETOOL_CHORE_EDIT	106
TRANSLATETOOL_CHORE_LANGUAGE_ADD	106
TRANSLATETOOL_CHORE_LANGUAGE_EDIT	106
TRANSLATETOOL_CHORE_LANGUAGE_SAVE	106
translatetool.class.php	106
<i>/program/lib/translatetool.class.php - taking care of language translations</i>	
task_logview()	104
<i>quick and dirty logfile viewer</i>	
TASK_TOOLS_INTRO	103
TASK_TRANSLATETOOL	103
task_backuptool()	104
<i>show an introductory text for backup tool OR stream a ZIP-file to the browser</i>	
TRANSLATETOOL_CHORE_LANGUAGE_SAVE_NEW	106
TRANSLATETOOL_CHORE_OVERVIEW	106
TASK_UPDATE_OVERVIEW	107
TASK_UPDATE_THEME	107
t()	133
<i>translation of phrases via a function with a very short name</i>	
Theme	321
<i>Methods to access properties of a theme</i>	
TASK_UPDATE_MODULE	107
TASK_UPDATE_CORE	107
TRANSLATETOOL_CHORE_SAVE	106
TRANSLATETOOL_PARAM_DOMAIN	106
TRANSLATETOOL_PARAM_LANGUAGE_KEY	106
<i>This parameter identifies the language.</i>	
TASK_LOGVIEW	103
TASK_BACKUPTOOL	103
TASK_SAVE_NEWPAGE	98
TASK_SAVE_NEWSECTION	98

TASK_SAVE_NODE	98
TASK_SET_DEFAULT	98
TASK_SAVE_CONTENT	98
TASK_PAGE_PREVIEW	98
TASK_NODE_EDIT	97
TASK_NODE_EDIT_ADVANCED	97
TASK_NODE_EDIT_CONTENT	98
TASK_SUBTREE_COLLAPSE	98
TASK_SUBTREE_EXPAND	98
theme.class.php	100
<i>/program/lib/theme.class.php - taking care of themes</i>	
themelib.php	101
<i>/program/lib/themelib.php - theme factory</i>	
theme_factory()	101
<i>manufacture a theme object</i>	
toolslib.php	103
<i>/program/lib/toolslib.php - tools</i>	
TREE_VIEW_MINIMAL	98
TREE_VIEW_MAXIMAL	98
TASK_TREEVIEW	98
TASK_TREEVIEW_SET	98
TREE_VIEW_CUSTOM	98
TASK_ACCOUNTS	19
<i>default selection for account manager: show introduction + links to users and groups</i>	

U

UserManager::get_fname()	359
<i>shorthand for the first readable name in a dialogdef item</i>	
UserManager::get_editor_names()	359
<i>prepare a list of available editors</i>	
UserManager::get_dialogdef_edit_user()	359
<i>construct the edit user dialog</i>	
UserManager::get_dialogdef_add_usergroup()	358
<i>construct a dialogdef for selecting a group/capacity</i>	
UserManager::get_icon_delete()	359
<i>construct a clickable icon to delete this user</i>	
UserManager::get_icon_edit()	360
<i>construct a clickable icon to edit the properties of this user</i>	
UserManager::get_user_names()	362
<i>shortcut to retrieve the username and full name of the selected user</i>	
UserManager::get_options_available_groups_capacities()	361
<i>construct a list of groups still available for this user</i>	
UserManager::get_num_user_records()	361
<i>calculate the total number of users in a specific group</i>	
UserManager::get_icon_groupdelete()	360
<i>construct a clickable icon to delete a membership from this user</i>	
UserManager::get_dialogdef_add_user()	358
<i>construct the add userdialog</i>	
UserManager::delete_user_acl()	358
<i>remove all records relating to a single acl_id from various acl-tables</i>	
UserManager	356

User management	
Useraccount::where_acl_id()	356
a convenient routine to construct a selection of acls	
Useraccount::is_admin_pagemanager()	356
determine whether the user has administrator privilege for pagemanager	
Useraccount::is_admin()	355
determine whether the user has administrator privilege	
UserManager::\$output	357
UserManager::\$show_parent_menu	357
UserManager::calc_acl_id()	358
UserManager::a_params()	358
shorthand for the anchor parameters that lead to the user manager	
UserManager::areas_expand_collapse()	357
manipulate the current state if indicator(s) for 'open' and 'closed' areas	
UserManager::get_user_records()	362
retrieve (a selection of) all user records from the database	
UserManager::has_job_permission()	363
determine whether a user has permissions for a particular job	
UserManager::user_groupsave()	368
save the new group/capacity for the selected user	
UserManager::user_groups()	367
present an overview of group memberships for the specified user	
UserManager::user_groupdelete()	367
end the group membership for the selected user	
UserManager::user_groupadd()	367
present 'add membership' dialog	
UserManager::user_intranet()	368
show a dialog for modifying intranet permissions for a user	
UserManager::user_pagemanager()	368
show a dialog for modifying page manager permissions for a user	
UserManager::user_save_basic()	369
save basic properties of user account	
UserManager::user_savenew()	369
save a new user to the database	
UserManager::user_save()	369
save edited user data to the database	
UserManager::user_edit()	367
present an 'edit user' dialog filled with existing data	
UserManager::user_delete()	366
delete a user after confirmation	
UserManager::show_menu_overview()	364
display a menu showing groups of users (if any) + corresponding breadcrumb trail	
UserManager::show_breadcrumbs_user()	363
display breadcrumb trail that leads to the edit user dialog	
UserManager::show_breadcrumbs_overview()	363
display breadcrumb trail that leads to users overview screen	
UserManager::show_breadcrumbs_adduser()	363
display breadcrumb trail that leads to the add new user dialog	
UserManager::show_menu_user()	365
show the user menu with current option highlighted	
UserManager::show_parent_menu()	365
UserManager::user_admin()	366
show a dialog for modifying admin permissions for a user	

UserManager::user_add()	366
<i>present 'add user' dialog where the user can enter minimal properties for a new user</i>	
UserManager::users_overview()	365
<i>display a list of existing users and an option to add a user</i>	
Useraccount::has_site_permissions()	355
<i>determine user's permissions for the site-level</i>	
Useraccount::has_node_permissions()	355
<i>determine user's permissions for a node within an area</i>	
useraccount.class.php	111
<i>/program/lib/useraccount.class.php - taking care of useraccounts</i>	
update_theme()	110
<i>call the theme-specific upgrade routine</i>	
update_status_update()	110
<i>return an anchor tag with link to the specific update function</i>	
update_module()	109
<i>call the module-specific upgrade routine</i>	
usermanager.class.php	114
<i>/program/lib/usermanager.class.php - taking care of user management</i>	
update_statistics()	151
<i>update all statistics for the view of page \$node_id</i>	
Useraccount::\$acls	350
Useraccount	347
<i>Methods to access properties of the account of the logged in user</i>	
update_view_count()	152
<i>update the view count for page \$node_id</i>	
update_core_version()	109
<i>record the specified version number in the config table AND in \$CFG->version</i>	
update_core_2011020100()	109
<i>perform actual update to version 2011020100</i>	
USERMANAGER_DIALOG_INTRANET	20
USERMANAGER_DIALOG_EDIT	20
USERMANAGER_DIALOG_DELETE	20
USERMANAGER_DIALOG_ADMIN	20
USERMANAGER_DIALOG_PAGEMANAGER	20
updatelib.php	107
<i>/program/lib/updatelib.php - update wizard</i>	
update_core_2010122100()	109
<i>perform actual update to version 2010122100</i>	
update_core_2010120800()	108
<i>perform actual update to version 2010120800</i>	
update_core()	108
<i>update the core version in the database to the version in the version.php file (the 'manifest' version)</i>	
Useraccount::\$acls_areas	350
Useraccount::\$acls_modules	350
Useraccount::\$username	353
Useraccount::\$related_acls	353
Useraccount::\$properties	353
Useraccount::\$path	352
Useraccount::\$user_id	353
Useraccount::fetch_acls_from_table()	354
<i>retrieve acl-data from table into a sparse array</i>	
Useraccount::has_job_permissions()	355

<i>determine user's permissions for a job</i>	
Useraccount::has_intranet_permissions()	354
<i>determine user's permissions for an intranet area</i>	
Useraccount::has_area_permissions()	354
<i>determine user's permissions for an area</i>	
Useraccount::\$language_key	352
Useraccount::\$is_logged_in	352
Useraccount::\$acl_id	351
Useraccount::\$sacIs_nodes	351
Useraccount::\$sacIs_modules_nodes	350
Useraccount::\$sacIs_modules_areas	350
Useraccount::\$area_permissions_from_nodes	351
Useraccount::\$editor	351
Useraccount::\$high_visibility	352
Useraccount::\$full_name	352
Useraccount::\$email	351
USERMANAGER_DIALOG_ADD	20

V

version.php	154
<i>'version.php' defines internal and external version numbers</i>	
valid_datetime()	59
<i>check validity of date, time or datetime</i>	

W

WAS_RELEASE	155
<i>The external version number, like 1.0 or 1.0.0</i>	
WAS_ORIGINAL	155
<i>A boolean flag indicating this is either the original (TRUE) or a modified (FALSE) version of Website@School</i>	
WAS_RELEASE_DATE	155
<i>Date of distribution file generation in ISO 8601 format: yyyy-mm-dd OR yyyy-mm-ddThh:mm:ss+0000</i>	
WAS_VERSION	155
<i>The internal version number, like 2008012873 or 2008020100 (31 bits will work until the year 2147)</i>	
WASENTRY	381
<i>Valid entry points define WASENTRY; prevents direct access to include()'s.</i>	
waslib.php	115
<i>/program/lib/waslib.php - core functions</i>	
was_logout()	91
<i>end a session (logout the user) and maybe redirect</i>	
was_version_check()	11
<i>check version of PHP-files against version stored in database</i>	
was.php	15
<i>/program/languages/en/was.php - generic translated messages</i>	
was.php	18
<i>/program/languages/nl/was.php - generic translated messages (Dutch)</i>	
was_login()	90

execute the selected login procedure

WASENTRY	8
<i>Valid entry points define WASENTRY; prevents direct access to include()'s.</i>	

Z

Zip::CloseZip()	374
<i>finish the ZIP-archive by outputting the central directory and closing output</i>	
Zip::dos_time_date()	374
<i>construct an MS-DOS time and date based on unix timestamp</i>	
Zip::AddFile()	374
<i>add the contents of an existing file to the current ZIP-archive</i>	
Zip::AddData()	373
<i>add data to the current ZIP-archive</i>	
Zip::\$zip_type	373
Zip::make_suitable_filename()	375
<i>construct a suitable filename for use in ZIP-archive</i>	
Zip::OpenZipbuffer()	375
<i>prepare the user supplied buffer for subsequent ZIP-archive data</i>	
Zip::zip_add_directories()	377
<i>workhorse function to add 0, 1 or more directories to the current ZIP-archive</i>	
Zip::zip_error()	377
<i>add an error message to the list of error messages</i>	
Zip::zip_add_data()	376
<i>workhorse function to add data to the current ZIP-archive</i>	
Zip::OpenZipstream()	376
<i>start with a stream (direct output) indicating an application/zip type of content</i>	
Zip::OpenZipfile()	376
<i>open a file for subsequent output of ZIP-archive</i>	
Zip::\$zip_path	373
Zip::\$zip_filehandle	373
ZIP_TYPE_STREAM	135
Zip	370
<i>Create simple and compatible ZIP-archives</i>	
ZIP_TYPE_NONE	135
ZIP_TYPE_FILE	135
ZIP_TYPE_BUFFER	135
Zip::\$central_directory	372
Zip::\$Error	372
Zip::\$zip_comment	373
Zip::\$zip_buffer	372
Zip::\$offset	372
Zip::\$no_name_files	372
zip.class.php	135
<i>/program/lib/zip.class.php - create simple ZIP-archives</i>	