

patUser



Contents

Package patUser Procedural Elements	2
example.php	2
example2.php	3
example_authHandler.php	4
example_cryptFunction.php	5
Function myCrypter	5
example_getPermissions.php	6
example_modifyUser.php	7
example_patConfiguration.php	8
patUser.php	9
Define patUSER_ALREADY_JOINED_GROUP	9
Define patUSER_COULD_NOT_IDENTIFY_LINE	9
Define patUSER_DELETE_NOT_ALLOWED	9
Define patUSER_GROUP_ALREADY_EXISTS	9
Define patUSER_INSERT_NOT_ALLOWED	9
Define patUSER_LOGIN_DISABLED	9
Define patUSER_NEED_GID	9
Define patUSER_NEED_GROUPNAME	9
Define patUSER_NEED_ID	10
Define patUSER_NEED_ID_TYPE	10
Define patUSER_NEED_PASSWD	10
Define patUSER_NEED_UID	10
Define patUSER_NEED_USERNAME	10
Define patUSER_NOT_IN_GROUP	10
Define patUSER_NO_DATA_CHANGED	10
Define patUSER_NO_DATA_GIVEN	10
Define patUSER_NO_DB_RESULT	10
Define patUSER_NO_GROUP_FOUND	10
Define patUSER_NO_PRIMARY_FOUND	10
Define patUSER_NO_UNIQUE_GROUP_FOUND	10
Define patUSER_NO_UNIQUE_PRIMARY_FOUND	10
Define patUSER_NO_UNIQUE_USER_FOUND	11
Define patUSER_NO_USER_FOUND	11
Define patUSER_PASSWD_MISMATCH	11
Define patUSER_TABLE_DOES_NOT_EXIST	11
Define patUSER_USER_ALREADY_EXISTS	11
prepend.php	12
Package patUser Classes	13
Class myClass	13
Method crypter	13
Class patUser	14
Var \$actionVar	14
Var \$authenticated	14

Var \$authFields	14
Var \$authHandler	14
Var \$authTable	15
Var \$cryptFunction	15
Var \$cryptSalt	15
Var \$dbcs	15
Var \$errorMessagees	16
Var \$errors	16
Var \$fieldLocs	16
Var \$groupFields	16
Var \$groupTable	16
Var \$ignoreVars	17
Var \$loginTemplate	17
Var \$maxLoginAttempts	17
Var \$permFields	17
Var \$perms	17
Var \$permsConv	17
Var \$permTable	17
Var \$realm	18
Var \$relFields	18
Var \$relTable	18
Var \$sessionVar	18
Var \$stats	18
Var \$statsUpdated	18
Var \$systemVars	18
Var \$tables	19
Var \$tpl	19
Var \$unauthorizedTemplate	19
Var \$unauthorizedURL	19
Var \$usePermissions	19
Var \$userIdSequence	19
Var \$useSessions	20
Var \$useTemplate	20
Constructor <u>construct</u>	20
Constructor <u>patUser</u>	20
Method <u>addDbc</u>	21
Method <u>addFieldLocation</u>	21
Method <u>addGroup</u>	21
Method <u>addPermission</u>	22
Method <u>addStats</u>	22
Method <u>addTable</u>	23
Method <u>addUser</u>	23
Method <u>addUserToGroup</u>	24
Method <u>authenticate</u>	24
Method <u>clearErrors</u>	24
Method <u>clearSessionValue</u>	25
Method <u>convertOptionsToSql</u>	25
Method <u>countTableEntries</u>	25
Method <u>deleteAllPermissions</u>	26

Method deleteGroup	26
Method deletePermission	26
Method deleteUser	27
Method encryptPasswd	27
Method getAllErrorCodes	27
Method getAllErrorMessages	28
Method getAllErrors	28
Method getAuthFields	28
Method getDbc	28
Method getField	28
Method getGroupData	29
Method getGroupFields	29
Method getGroups	29
Method getHistory	30
Method getJoinedGroups	30
Method getLastError	30
Method getLastErrorCode	31
Method getLastErrorMessage	31
Method getPermissions	31
Method getPermTable	31
Method getPrimaryValue	32
Method getSessionValue	32
Method getTableDef	33
Method getTableInfo	33
Method getUid	33
Method getUserData	33
Method getUsers	34
Method getUsersInGroup	34
Method goHistory	35
Method hasPermission	35
Method identifyGroup	35
Method identifyUser	36
Method isAuthenticated	36
Method isMemberOfGroup	36
Method issetSessionValue	37
Method keepHistory	37
Method logOut	37
Method modifyGroup	38
Method modifyUser	38
Method removeLastError	39
Method removeUserFromGroup	39
Method requireAuthentication	39
Method searchUsers	40
Method sendQuery	40
Method setAuthDbc	41
Method setAuthDsn	41
Method setAuthFields	42
Method setAuthHandler	42
Method setAuthTable	42

Method setCryptFunction	43
Method setError	43
Method setGroupFields	43
Method setGroupRelFields	44
Method setGroupRelTable	44
Method setGroupTable	45
Method setLoginTemplate	45
Method setMaxLoginAttempts	45
Method setPermFields	46
Method setPermTable	46
Method setRealm	47
Method setTemplate	47
Method setUnauthorizedTemplate	47
Method setUnauthorizedURL	48
Method storeSessionValue	48
Method translateErrorCode	49
Method updateStats	49
Class testHandler	49
Method patUserGetAuthData	49
Method patUserSetErrors	50
Method patUserSetRealm	50
Method patUserSetUid	50

[Package patConfiguration Classes](#) 51

Class patConfiguration	51
Var \$cacheDir	51
Var \$conf	51
Var \$currentConf	51
Var \$data	52
Var \$defaultTypes	52
Var \$extensions	52
Var \$externalFiles	52
Var \$includeDir	52
Var \$nsStack	52
Var \$path	52
Var \$valDepth	52
Var \$valStack	53
Var \$xmlFiles	53
Var \$xmlSpecialchars	53
Method addExtension	53
Method appendData	53
Method characterData	54
Method clearConfigValue	54
Method createParser	54
Method endElement	54
Method externalEntity	54
Method getConfigValue	55
Method loadCachedConfig	55
Method niceDie	55
Method parseConfigFile	56

Method parseXMLFile	56
Method replaceXMLSpecialchars	56
Method setCacheDir	56
Method setConfigDir	57
Method setConfigValue	57
Method setConfigValues	58
Method setDefaultTypes	58
Method setIncludeDir	58
Method startElement	59
Method writeConfigFile	59
Package patTemplate Procedural Elements	61
patTemplate.php	61
Define patTEMPLATE_TAG_END	61
Define patTEMPLATE_TAG_START	61
Define patTEMPLATE_TYPE_CONDITION	61
Define patTEMPLATE_TYPE_ODDEVEN	61
Define patTEMPLATE_TYPE_SIMPLECONDITION	61
Define patTEMPLATE_TYPE_STANDARD	62
Package patTemplate Classes	63
Class patTemplate	63
Constructor patTemplate	63
Method addGlobalVar	64
Method addGlobalVars	64
Method addRows	65
Method addTemplate	65
Method addTemplates	65
Method addVar	66
Method addVars	66
Method clearAllTemplates	67
Method clearAttribute	67
Method clearTemplate	67
Method displayParsedTemplate	68
Method dump	68
Method exists	69
Method getAttribute	69
Method getParsedTemplate	69
Method getVar	70
Method parseTemplate	70
Method readTemplatesFromFile	71
Method setAttribute	71
Method setAttributes	71
Method setBasedir	72
Method setTags	72
Method setType	73
Appendices	74
Appendix A - Class Trees	75
patConfiguration	75
patUser	75

patTemplate	75
Appendix D - Todo List	76

Package patUser Procedural Elements

example.php

- **Package** patUser

include ["append.php"](#)*[line 50]*

include ["prepend.php"](#)*[line 5]*

patUser example

example2.php

- **Package** patUser

include ["append.php"](#)*[line 11]*

include ["prepend.php"](#)*[line 5]*

patUser example

example_authHandler.php

- **Package** patUser

include ["append.php"](#)*[line 97]*

include ["prepend.php"](#)*[line 64]*

patUser example

example_cryptFunction.php

example_cryptFunction
\$Id%

- **Package** patUser
- **Copyright** 2003 Metrix Internet Design GmbH <http://www.metrix.de>

include ["prepend.php"](#) [line 59]

```
*****  
start with patUser here  
*****
```

*

include ["append.php"](#) [line 97]

string function myCrypter(\$password, \$salt) [line 48]

Function Parameters:

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

demo crypt function

This is a "global" function. this is no real crypt function! There is no crypt-algorithm included!

example_getPermissions.php

- **Package** patUser

include ["append.php"](#)*[line 68]*

include ["prepend.php"](#)*[line 5]*

patUser example

example_modifyUser.php

- **Package** patUser

include ["append.php"](#)*[line 70]*

include ["prepend.php"](#)*[line 5]*

patUser example

example_patConfiguration.php

- **Package** patUser

include_once ["include/patUser.php"](#)*[line 41]*

include_once "DB.php" *[line 29]*

include_once ["include/patTemplate.php"](#)*[line 24]*

include_once ["include/patConfiguration.php"](#)*[line 15]*

example for patListShare

patUser.php

Powerfull user/group/permission management class based on databases.

- **Package** patUser
- **TODO** check user
- **Author** Stephan Schmidt < schst@php-tools.net>
- **Author** gERD Schaufelberger < gerd@php-tools.net>

\$Id: patUser.php 5 2005-03-02 10:51:24Z argh \$

patUSER_ALREADY_JOINED_GROUP = 1050 *[line 123]*
error code: user already is in group

patUSER_COULD_NOT_IDENTIFY_LINE = 110 *[line 73]*
error code: could not identify line in table

patUSER_DELETE_NOT_ALLOWED = 121 *[line 83]*
error code: delete is not allowed

patUSER_GROUP_ALREADY_EXISTS = 1012 *[line 113]*
error code: group already exists (when adding a group)

patUSER_INSERT_NOT_ALLOWED = 120 *[line 78]*
error code: insert is not allowed

patUSER_LOGIN_DISABLED = 14 *[line 48]*
error code: login for this user was diabled

patUSER_NEED_GID = 1020 *[line 118]*
error code: function requires group id

patUSER_NEED_GROUPNAME = 1010 *[line 108]*
error code: function requires a group name

patUSER_NEED_ID = 1060 *[line 133]*

error code: function requires user or group id

patUSER_NEED_ID_TYPE = 1061 *[line 138]*

error code: function requires type of supplied id (user or group)

patUSER_NEED_PASSWD = 11 *[line 33]*

error code: function requires a password

patUSER_NEED_UID = 20 *[line 53]*

error code: function requires a user id

patUSER_NEED_USERNAME = 10 *[line 28]*

error code: function requires a user name

patUSER_NOT_IN_GROUP = 1051 *[line 128]*

error code: user is not in group

patUSER_NO_DATA_CHANGED = 130 *[line 88]*

error code: no data was changed (affected rows = 0)

patUSER_NO_DATA_GIVEN = 30 *[line 58]*

error code: function requires data

patUSER_NO_DB_RESULT = 2000 *[line 143]*

error code: query had no result

patUSER_NO_GROUP_FOUND = 1001 *[line 98]*

error code: no group matched the query

patUSER_NO_PRIMARY_FOUND = 100 *[line 63]*

error code: no primary key was found

patUSER_NO_UNIQUE_GROUP_FOUND = 1002 *[line 103]*

error code: no unique group matched the query

patUSER_NO_UNIQUE_PRIMARY_FOUND = 101 *[line 68]*

error code: data matched more than one row

patUSER_NO_UNIQUE_USER_FOUND = 2 *[line 23]*
error code: more than one user matror the que y

patUSER_NO_USER_FOUND = 1 *[line 18]*
error code: no user matched the query

patUSER_PASSWD_MISMATCH = 13 *[line 43]*
error code: password incorrect

patUSER_TABLE_DOES_NOT_EXIST = 140 *[line 93]*
error code: table does not exist

patUSER_USER_ALREADY_EXISTS = 12 *[line 38]*
error code: user already exists

prepend.php

- **Package** patUser

include_once **"DB.php"** [*line 25*]

include_once ["include/patUser.php"](#)[*line 17*]

include_once ["include/patTemplate.php"](#)[*line 11*]

patUser prepend

Package patUser Classes

Class myClass

[line 18]

demo class for external crypt functions

- **Package** patUser

string function myClass::crypter(\$password, \$salt) [line 31]

Function Parameters:

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

example crypt inside objects function

This is a member function of myClass this is no real crypt function! There is no crypt-algorithm included!

- **Access** public

Class patUser

[line 161]

user management class

patUser is a user management class, that helps you with authentication, groups and permission. Furthermore it is useful to manage data (stored databases) related to users and groups. Also patUser provides user statistics.

CAUTION: This version is based on PEAR:DB patDbc will no longer supported! - please switch to PEAR:DB patUser 2.1.x is the last version that supports patDbc.

- **Package** patUser
- **Version** 2.2.4
- **Author** Stephan Schmidt < schst@php-tools.net>
- **Author** gERD Schaufelberger < gerd@php-tools.net>

patUser::\$actionVar

```
string = "patUserAction" [line 364]
```

name of the global variable that indicates the action in the request

patUser::\$authenticated

```
bool = false [line 242]
```

state of user/session: authenticated or not

patUser::\$authFields

```
array = array( "primary" => "uid",  
              "username" => "username",  
              "passwd" => "passwd" ) [line 248]
```

Fieldnames in the authentication table

patUser::\$authHandler

```
mixed = false [line 358]
```

authentication handler object (false if no handler is used)

The authentication handler object will be used by `{@see requireAuthentication()}` to receive data used for authentication. Therefore the handler-object must implement a method called: `patUserGetAuthData()`.

Moreover `patUser` supports other optional methods of the auth handler object:

- `patUserSetUid`: sends the user-id if user is logged in.
- `patUserSetRealm`: send the realm-string when no user is logged in
- `patUserSetErrors`: sends `patUser-errors` if login failed

- See [patUser::setAuthHandler\(\)](#), `requireAuthentication()`, [patUser::\\$useTemplate](#)

`patUser::$authTable`

```
string = "users" [line 236]
```

Table that stores the authentication data

`patUser::$cryptFunction`

```
string = false [line 418]
```

name of function, that should be used for passwd encryption

`patUser::$cryptSalt`

```
string = null [line 424]
```

salt string for stronger encryption

`patUser::$dbcs`

```
array = array() [line 388]
```

all dbcs

`patUser::$errorMessages`

```
array = array(  
    1 => "No user found.",  
    2 => "No unique user found.",
```

```

10 => "Username is required.",
11 => "Password is required.",
12 => "User already exists.",
13 => "Passwords do not match.",
14 => "You are not allowed to login.",
20 => "User Id is required.",
30 => "Data is needed.",
100 => "No primary key value found.",
101 => "No unique primary key value found.",
110 => "Dataset could not be identified.",
120 => "Insert is not allowed.",
121 => "Delete is not allowed.",
130 => "Data was not changed.",
140 => "Table does not exist.",
1001 => "No group found.",
1002 => "No unique group found.",
1010 => "Name of group is required.",
1012 => "Group already exists.",
1020 => "Group Id is required.",
1050 => "User already is in group.",
1051 => "User is not in group.",
1060 => "Need user or group id.",
1061 => "No id type specified.",
2000 => "No database result id returned."
) [line 180]

```

error messages

patUser::\$errors

```
array = array() [line 400]
```

all codes of the errors that happened while processing

patUser::\$fieldLocs

```
array = array() [line 230]
```

locations (table/fieldname) of fields

patUser::\$groupFields

```
array = array( "primary" => "gid",
               "name" => "name" ) [line 262]
```

Fieldnames in the group table

patUser::\$groupTable

```
string = "groups" [line 256]
```

Table that stores the group data

patUser::\$ignoreVars

```
array = array( "patUserAction" ) [line 406]
```

variable names that should be ignored by getSelfUrl()

patUser::\$loginTemplate

```
string = "patUserLogin.tmpl" [line 370]
```

filename of the template used for login

patUser::\$maxLoginAttempts

```
integer = 0 [line 218]
```

maximum login attempts for a session

patUser::\$permFields

```
array = array( "id" => "id",  
              "id_type" => "id_type",  
              "perms" => "perms" ) [line 288]
```

Fieldnames in the permissions table

patUser::\$perms

```
array = array( 1 => "read",  
              2 => "delete",  
              4 => "modify",  
              8 => "add" ) [line 296]
```

Possible permissions

patUser::\$permsConv

```
array = array() [line 304]
```

table to convert permissions

patUser::\$permTable

```
string = "permissions" [line 282]
```

Table that stores the permissions

patUser::\$realm

```
string = "patUser protected area" [line 212]
```

default realm for HTTP authentication

patUser::\$relFields

```
array = array( "uid" => "uid",  
              "gid" => "gid" ) [line 275]
```

Fieldnames of the user-group relation table

patUser::\$relTable

```
string = "usergroups" [line 269]
```

Table that stores the user - group relations

patUser::\$sessionVar

```
string = "patUserData" [line 329]
```

name of the global variable used for sessions

patUser::\$stats

```
array = array() [line 310]
```

all statistic options

patUser::\$statsUpdated

```
string = false [line 412]
```

whether user stats have been updated.

patUser::\$systemVars

```
array = array(  
    "appName" => "patUser",  
    "appVersion" => "2.2.4",  
    "author" => array(  
        "Stephan Schmidt <schst@php-tools.net>",  
        "Gerd Schaufelberger <gerd@php-tools.net>"  
    )) [line 167]
```

information about the project

patUser::\$tables

array = array() [line 394]

all tables that are used

patUser::\$tmpl

boolean = false [line 341]

variable to store the patTemplate object (false if no template is used)

patUser::\$unauthorizedTemplate

string = "patUserUnauthorized.tpl" [line 376]

filename of the template used for unauthorized users

patUser::\$unauthorizedURL

string = false [line 382]

URL to redirect unauthorized users to

patUser::\$usePermissions

boolean = false [line 323]

flag to indicate whether permissions are available

- See [patUser::setPermTable\(\)](#)

patUser::\$userIdSequence

string = "patUserSequence" [line 335]

name of the sequence for user ids

patUser::\$useSessions

boolean = *false* [*line 316*]

flag to indicate whether sessions are used

patUser::\$useTemplate

string = *false* [*line 224*]

flag to indicate, whether template is used for output

Constructor function patUser::__construct([\$useSessions = true], [\$sessionVar = "patUserData"], [\$userIdSequence = "patUserSequence"]) [*line 452*]

Function Parameters:

- *boolean* **\$useSessions** flag to indicate that sessions should be used
- *string* **\$sessionVar** name of the session var used for sessions
- *string* **\$userIdSequence** name of the sequence for retrieving the next user id

create new user object

constructor of patUser

- **Access** public

Constructor function patUser::patUser([\$useSessions = true], [\$sessionVar = "patUserData"], [\$userIdSequence = "patUserSequence"]) [*line 437*]

Function Parameters:

- *boolean* **\$useSessions** flag to indicate that sessions should be used
- *string* **\$sessionVar** name of the session var used for sessions
- *string* **\$userIdSequence** name of the sequence for retrieving the next user id

create new user object

constructor for use with PHP4

- See [patUser::__construct\(\)](#)
- Access public

boolean function patUser::addDbc(\$name, \$dsn, [\$persistent = false]) [*line 1633*]

Function Parameters:

- *string* **\$name** unique name to access the dbc
- *mixed* **\$dsn** datasource name or PEAR:DB object
- *boolean* **\$persistent** flag to indicate whether a persistent connection should be established

add a dsn or dbc-object

you can use as many databases as you like to access data

- See [patUser::\\$dbcs](#), [patUser::addDbc\(\)](#)
- Access public

function patUser::addFieldLocation(\$name, [\$table = "authTable"], [\$field = ""]) [*line 864*]

Function Parameters:

- *string* **\$name** name of the field
- *string* **\$table** name of the table that stores the field (use "authTable" if its stored in the authtable)
- *string* **\$field** name of the field in the table (leave of if identical with name)

set the location of a field

- Access public

int function patUser::addGroup([\$data = array()]) [*line 2534*]

Function Parameters:

- *array* **\$data** date for the group

adds a group

- **Access** public

bool function patUser::addPermission(\$perm) [*line 3001*]

Function Parameters:

- *array* **\$perm** array containing the permission(s) to be added

add permission(s)

- **Access** public

function patUser::addStats(\$statistic, [\$field = ""]) [*line 848*]

Function Parameters:

- *string* **\$statistic** name of the statistic that should be tracked
- *string* **\$field** name of the field that should store the stats (if it differs from the name)

add a statistic option

stats include: first_login, last_login, count_logins, count_pages, time_online

- See [patUser::\\$stats](#)
- **Access** public

function patUser::addTable(\$name, \$tabledef) [*line 1672*]

Function Parameters:

- *string* **\$name** internal name of the table
- *array* **\$tabledef** array containing information about the table

add table that is used to store user data

values for tabledefs:

- **foreign** (required) name of the field, that stores the foreign key
- **primary** (optional) only needed when more than one line for each user is stored in the table
- **table** (optional) name of table in database
- **dbc** (optional) dbc to use
- **minentries** (optional) minimum amount of entries in this table
- **maxentries** (optional) maximum amount of entries in this table

- **Access** public

int function patUser::addUser(\$authData, [\$login = true]) [*line 1374*]

Function Parameters:

- *array* **\$authData** data for the user that will be stored in the authtable (compared with authFields)
- *boolean* **\$login** automatically login after creation

adds a user

create a new user and store it basic-authentication data.

- See [patUser::\\$authTable](#), [patUser::\\$authFields](#)
- **Access** public

bool function patUser::addUserToGroup(\$relation) [*line 2675*]

Function Parameters:

- *array* **\$relation** user id and group id

add a user to a group

- **Access** public

mixed function patUser::authenticate([\$data = array()]) [*line 909*]

Function Parameters:

- *array* **\$data** array containing data for authentication (username, passwd)

authenticate a user

This method is used to log in. This method uses the \$data-parameter and tries to find a matching user. Furthermore it checks if the user is allowed to login (uses the "nologin" field in the auth-table). If the user was logged in successfully, the statistics will be updated (if the statistic-function is enabled).

- **See** [patUser::isAuthenticated\(\)](#), [patUser::requireAuthentication\(\)](#)
- **Access** public

function patUser::clearErrors() [*line 3647*]

clear all errors

- **Access** public

bool function patUser::clearSessionValue(\$name) [*line 1474*]

Function Parameters:

- *string* **\$name** name of the value to clear

clear a value in the session

may only be used if patUser was called with \$useSessions = true

- See [patUser::\\$useSessions](#), [patUser::storeSessionValue\(\)](#), [patUser::getSessionValue\(\)](#)
- Access public

string function patUser::convertOptionsToSql([\$options = array()]) [*line 3557*]

Function Parameters:

- *array* **\$options** assoc array containing all options

convert options to sql

access private

function patUser::countTableEntries(\$table, [\$uid = 0]) [*line 2197*]

Function Parameters:

- *string* **\$table** name of the table
- *integer* **\$uid** user id (logged in user is used if none is given)

count amount of entries for a user in a table

- Access public

bool function patUser::deleteAllPermissions(\$clause) [*line 3094*]

Function Parameters:

- *array* **\$clause** array containing conditions for the where clause

delete all permissions without setting new ones

- **Access** public

function patUser::deleteGroup([\$groupdata = array()]) [*line 2580*]

Function Parameters:

- *array* **\$groupdata** data used to identify a group

delete an existing group

- **Access** public

bool function patUser::deletePermission(\$perm) [*line 3026*]

Function Parameters:

- *array* **\$perm** array containing the permission(s) to be deleted

delete permission(s)

- **Access** public

function patUser::deleteUser([\$options = array()]) [*line 2026*]

Function Parameters:

- *array* **\$options** options for the deletion

delete (data of) an existing user

- **Access** public

string function patUser::encryptPasswd(\$passwd, \$salt) [*line 3769*]

Function Parameters:

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

encrypt a password using the specified encryption function

- See [patUser::setCryptFunction\(\)](#)
- **Access** public

array function patUser::getAllErrorCodes() [*line 3692*]

get all error codes

- **Access** public

array function patUser::getAllErrorMessages() [line 3703]

get all error messages

- Access public

array function patUser::getAllErrors() [line 3718]

get all error codes AND messages

- Access public

array function patUser::getAuthFields() [line 626]

get fieldnames that contain auth info

- See [patUser::setAuthFields\(\)](#), [patUser::\\$authFields](#)
- Access public

object patDbc function patUser::getDbc(\$table) [line 3610]

Function Parameters:

- *string* **\$table** name of the table

get the dbc object for a table

mixed function patUser::getField(\$field, [\$uid = 0]) [line 2355]

Function Parameters:

- *string* **\$field** name of the table

- *integer* **\$uid** user id, if no uid is given the authenticated user will be used

fetch the value of a field

- **Access** public

array function patUser::getGroupData([\$fields = array()], [\$clause = array()]) [*line 2422*]

Function Parameters:

- *array* **\$fields** fields to get
- *array* **\$clause** params to identify the group

fetch the group data from group table

- **Access** public

array function patUser::getGroupFields() [*line 674*]

get fieldnames that contain group info

This method usually is called during setup the patUser-object

- **See** [patUser::setGroupFields\(\)](#), [patUser::\\$groupFields](#)
- **Access** public

function patUser::getGroups([\$fields = array()], [\$clause = array()], [\$options = array()]) [*line 2405*]

Function Parameters:

- *array* **\$fields** array containing fieldnames to be fetched

- *array* **\$clause** array containing conditions for the where statement
- *array* **\$options** array containing misc options

get groups from the grouptable

- **Access** public

array function patUser::getHistory([\$pages = 0]) [*line 1582*]

Function Parameters:

- *integer* **\$pages** number of the last pages to get

get the history for the current user

- **Access** public

array function patUser::getJoinedGroups([\$options = array()]) [*line 2607*]

Function Parameters:

- *array* **\$options** several options (like orderby, limit, offset or uid)

get joined groups

- **Access** public

array function patUser::getLastError() [*line 3680*]

get the last error code AND message

- **Access** public

integer function patUser::getLastErrorCode() [*line 3658*]

get the last error code

- **Access** public

string function patUser::getLastErrorMessage() [*line 3669*]

get the last error message

- **Access** public

array function patUser::getPermissions([\$clause = array()], [\$type = "both"]) [*line 2817*]

Function Parameters:

- *array* **\$clause** params for the permissions (any fields in your permission table)
- *string* **\$type** get user or group permissions or both (both|user|group|all)

get all permissions of a user or all permissions

- **Access** public

string function patUser::getPermTable() [*line 734*]

get tablename that contains permissions

- See [patUser::setPermTable\(\)](#), [patUser::\\$permTable](#)
- Access public

function patUser::getPrimaryValue(\$uid, \$table, \$data) [*line 2153*]

Function Parameters:

- *int* **\$uid** user id
- *string* **\$table** name of the table
- *array* **\$data** date of the line to identify

get the value of the primary key in an additional table identifies a unique line in an additional table

- Access public

mixed function patUser::getSessionValue(\$name) [*line 1495*]

Function Parameters:

- *string* **\$name** name of the value to retrieve

get a value that was stored in the session

may only be used if patUser was called with \$useSessions = true

- See [patUser::\\$useSessions](#), [patUser::clearSessionValue\(\)](#), [patUser::storeSessionValue\(\)](#)
- Access public

array function patUser::getTableDef(\$name) [line 1687]

Function Parameters:

- *string* **\$name** internal name of the table

get table definitions

- **Access** public

array function patUser::getTableInfo(\$tablename) [line 3397]

Function Parameters:

- *string* **\$tablename** internal table name

get information about table fields like DESCRIBE [table]

- **Access** public

int function patUser::getUid() [line 994]

an easy way to receive the id of the current user.

return the user id of the current user

- **Access** public

array function patUser::getUserData([\$options = array()], [\$clause = array()]) [line 1702]

Function Parameters:

- *array* **\$options** several options, like table name, user id and fields to get
- *array* **\$clause** assoc array containing fields/values for the where statement

fetch the user data from a table

- **Access public**

function patUser::getUsers([\$fields = array()], [\$clause = array()], [\$options = array()]) [*line 1762*]

Function Parameters:

- *array* **\$fields** array containing fieldnames to be fetched
- *array* **\$clause** array containing conditions for the where statement
- *array* **\$options** array containing misc options

get users from any table

- **Access public**

array function patUser::getUsersInGroup(\$fields, [\$options = array()]) [*line 2638*]

Function Parameters:

- *array* **\$fields** fields to get for the users (only from authTable)
- *array* **\$options** several options (like orderby, limit, offset)

get users in group

- **Access public**

function patUser::goHistory([\$pages = -1]) [line 1608]

Function Parameters:

- *integer* **\$pages** number of the pages to go back (use negative values)

go back x pages in the history as Header("Location:...") is used, there must not be any output before this is called

- **Access public**

boolean function patUser::hasPermission([\$perm = array()], [\$mode = "all"], [\$includeGroups = true])
[line 2909]

Function Parameters:

- *array* **\$perm** permission(s) to be checked
- *string* **\$mode** all or any permission (all|any)
- *boolean* **\$includeGroups** should group permissions be checked,too?

check, whether user has a permission

- **Access public**

function patUser::identifyGroup(\$fields, [\$createError = true]) [line 2446]

Function Parameters:

- *array* **\$fields** several fields to identify the group
- *boolean* **\$createError** if set to false, no error will be added to errorlist if no group was found

identify a group by certain fields

- **Access** public

function patUser::identifyUser(\$options) [*line 1784*]

Function Parameters:

- *array* **\$options** several options to identify the user

identify a user by certain fields

- **Access** public

bool function patUser::isAuthenticated() [*line 966*]

check, whether a user is already authenticated

if auth is done via session, all important vars will be fetched.

- **Access** public

boolean function patUser::isMemberOfGroup(\$uid, \$gid) [*line 2796*]

Function Parameters:

- *int* **\$uid** user id
- *int* **\$gid** group id

check, if a user is in a group

- **Access** public

boolean function patUser::issetSessionValue(\$name) [*line 1515*]

Function Parameters:

- *string* **\$name** name of the value to retrieve

check, whether a value has been stored in the session **may only be used if patUser**
was called with \$useSessions = true

- See [patUser::\\$useSessions](#)
- **Access** public

bool function patUser::keepHistory(\$amount, [\$title = ""]) [*line 1536*]

Function Parameters:

- *int* **\$amount** size of the history
- *string* **\$title** title of the current page

let patUser create a history of visited pages
needs session support

- **Access** public

function patUser::logout([\$force = true]) [*line 1338*]

Function Parameters:

- *boolean* **\$force** if set to false, a logout screen will be displayed (will be implemented in future versions)

force log out of an authenticated user

- **Access** public

```
function patUser::modifyGroup($olddata, $newdata) [line 2486]
```

Function Parameters:

- *array* **\$olddata** old data of the group
- *array* **\$newdata** new data of the group

modify an existing group

- **Access** public

```
function patUser::modifyUser($data, [$options = array()]) [line 1850]
```

Function Parameters:

- *array* **\$data** new data for the user
- *array* **\$options** various options like mode (insert|update), uid and table

modify an existing user

- **Access** public

function patUser::removeLastError() [*line 3637*]

remove the last error from the list

- **Access** public

bool function patUser::removeUserFromGroup(\$relation) [*line 2724*]

Function Parameters:

- *array* **\$relation** user id and group id (use all to delete all users / groups)

delete user(s) from group(s)

- **Access** public

int function patUser::requireAuthentication([\$mode = "displayLogin"], [\$displayOnError = true]) [*line 1064*]

Function Parameters:

- *string* **\$mode** what should be done if user is not authenticated (displayLogin|callAuthHandler|exit)
- *boolean* **\$displayOnError** flag, that indicates, whether the form should again be displayed on error

require an authenticated user

Use this function, if you want to be sure, that a user is logged in. If there is no user yet, this method tries to get the authentication data (usually username and password). This can be controlled by the "mode"-parameter. Set mode to:

- "displayLogin": use patTemplate to display a login screen and search the POST-variables for username and password. If there is no user, the programme will exit. This is the default mode.
- "exit": This very simple mode just exits the script, if no user is logged in.
- "callAuthHandler": In this case patUser sets the realm in the authHandler

object and asks for the authentication data to login user. If the user could authenticated, patUser sets the uid in the authHandler, otherwise it sends the errors to the authHandler. If user was already authenticated, patUser just sends the uid to the authHandler.

- See [patUser::setAuthHandler\(\)](#)
- Access public

array function patUser::searchUsers(\$criterias, [\$options = array()]) [line 2231]

Function Parameters:

- array **\$criterias** see documentation
- array **\$options** see documentation

Search for users matching certain criterias

- Access public

object DB_Result function patUser::sendQuery(\$dbc, \$query) [line 3536]

Function Parameters:

- string **\$dbc** name of the database
- string **\$query** query

send query to any database can be used the use the internal dbcs in your own apps without knowing how the user object was configured

- **Access** public

boolean function patUser::setAuthDbc(\$dbc, [\$persistent = false]) [*line 562*]

Function Parameters:

- *mixed* **\$dbc** dsn-string or PEAR::DB object
- *boolean* **\$persistent** flag to indicate, whether a persistent connection should be established

set dbc that contains auth info

Set the database (PEAR:DB) object that will be used to gain access to the authentication data

- See [patUser::setAuthDsn\(\)](#)
- **Access** public

object function patUser::setAuthDsn(\$dsn, [\$persistent = false]) [*line 539*]

Function Parameters:

- *string* **\$dsn** datasource name for authorization database
- *boolean* **\$persistent** flag to indicate, whether a persistent connection should be established

set dsn that contains authentication information

This method is DEPRECATED use setAuthDbc() instead. The dsn describes the database connection which will be used to receive the authentication data. The dsn will be used by PEAR::DB

- See [patUser::setAuthDbc\(\)](#)
- **Deprecated** since version 2.2.3, please use setAuthDbc() instead
- **Access** public

function patUser::setAuthFields(\$fields) [*line 609*]

Function Parameters:

- array **\$fields** assoc array containing fieldnames of the authtable

set fieldnames that contain auth info

This method usually is called during setup the patUser-object

- See [patUser::setAuthTable\(\)](#), [patUser::\\$authFields](#)
- Access public

boolean function patUser::setAuthHandler(&\$handler, &\$authHandler) [*line 776*]

Function Parameters:

- object **&\$authHandler** Object that handles authentication data
- **&\$handler**

set authentication handler object

the authentication handler supplies patUser with authentication data and handles logins (failed and succeeded). This allows patUser to work without http-auth or patTemplate.

The authHandler must implement at least one method: patUserGetAuthData. This method will be called if the authentication data is needed. Optional patUser supports more methods {@see \$authHandler}.

- See [patUser::\\$authHandler](#), [patUser::\\$useTemplate](#)
- Access public

function patUser::setAuthTable(\$table) [*line 595*]

Function Parameters:

- *string* **\$table** name of the table that contains auth data

set tablename that contains auth info

This method usually is called during setup the patUser-object

- See [patUser::setAuthFields\(\)](#), [patUser::\\$authTable](#)
- Access public

```
function patUser::setCryptFunction([$function = "crypt"], [$salt = null], $cryptFunction) [line 882]
```

Function Parameters:

- *mixed* **\$cryptFunction** name of the encryption function or array containing object-reference and function name
- *string* **\$salt** salt value, default is null, if no salt parameter should \$see \$cryptSalt
\$cryptFunction crypt()
- **\$function**

set function that is used for passwd encryption

- Access public

```
function patUser::setError($error) [line 3627]
```

Function Parameters:

- *int* **\$error** error code

set error code

```
function patUser::setGroupFields($fields) [line 655]
```

Function Parameters:

- *array* **\$fields** assoc array containing fieldnames of the grouptable

set fieldnames that contain group info

This method usually is called during setup the patUser-object

- See [patUser::setGroupTable\(\)](#), [patUser::\\$groupFields](#)
- Access public

```
function patUser::setGroupRelFields($fields) [line 702]
```

Function Parameters:

- *array* **\$fields** assoc array containing fieldnames of the user group relation table

set fieldnames that are used in the user group relations

This method usually is called during setup the patUser-object

- See [patUser::setGroupRelTable\(\)](#), [patUser::\\$relFields](#)
- Access public

```
function patUser::setGroupRelTable($table) [line 688]
```

Function Parameters:

- *string* **\$table** name of the table that contains user group relations

set tablename that contains user group relations

This method usually is called during setup the patUser-object

- See [patUser::setGroupRelFields\(\)](#), [patUser::\\$relTable](#)
- Access public

function patUser::setGroupTable(\$table) [*line 641*]

Function Parameters:

- *string* **\$table** name of the table that contains group data

set tablename that contains group info

This method usually is called during setup the patUser-object

- See [patUser::setGroupFields\(\)](#), [patUser::\\$groupTable](#)
- Access public

function patUser::setLoginTemplate(\$tmplFile) [*line 815*]

Function Parameters:

- *string* **\$tmplFile** The login file (relative to the patTemplate base path)

Sets the name of the login template to use, to override the default 'patUserLogin.tpl'

Note: this only has any effect if you set a template object via [setTemplate\(\)](#).

- See [patUser::\\$loginTemplate](#)
- See [patUser::setTemplate\(\)](#)
- Access public

function patUser::setMaxLoginAttempts(\$maxLoginAttempts) [*line 509*]

Function Parameters:

- *integer* **\$maxLoginAttempts**

set maximum amount of login attempts

- **Access** public

function patUser::setPermFields(\$fields) [*line 749*]

Function Parameters:

- *array* **\$fields** assoc array containing fieldnames of the permission table

set fieldnames that are used in the permissions

This method usually is called during setup the patUser-object

- **See** `$setPermTable()`, [patUser::\\$permFields](#)
- **Access** public

function patUser::setPermTable(\$table) [*line 721*]

Function Parameters:

- *string* **\$table** name of the table that contains permissions

set tablename that contains permissions

This method usually is called during setup the patUser-object

- See [patUser::setPermFields\(\)](#), [patUser::\\$permTable](#)
- Access public

function patUser::setRealm(\$realm) [*line 498*]

Function Parameters:

- *string* **\$realm** authentication realm

set authentication realm

- Access public

function patUser::setTemplate(&\$tmpl) [*line 793*]

Function Parameters:

- *object* **patTemplate** **&\$tmpl** patTemplate Object

set template object the template object is used for displaying login / logout screens

- Access public

function patUser::setUnauthorizedTemplate(\$tmplFile) [*line 833*]

Function Parameters:

- *string* **\$tmplFile** The login file (relative to the patTemplate base path)

Sets the name of the template to use for unauthorized access, to override the default 'patUserUnauthorized.tmpl'

Note: this only has any effect if you set a template object via [setTemplate\(\)](#).

- See [patUser::setLoginTemplate\(\)](#)
- See [patUser::\\$unauthorizedTemplate](#)
- See [patUser::setTemplate\(\)](#)
- Access public

function patUser::setUnauthorizedURL(\$url) [*line 520*]

Function Parameters:

- *string* **\$url**

set URL to redirect unauthorized users to

- Access public

bool function patUser::storeSessionValue(\$name, \$val) [*line 1455*]

Function Parameters:

- *string* **\$name** name of the value to store
- *mixed* **\$val** value to store

store a value in the session

may only be used if patUser was called with \$useSessions = true

- See [patUser::\\$useSessions](#), [patUser::clearSessionValue\(\)](#), [patUser::getSessionValue\(\)](#)
- Access public

string function patUser::translateErrorCode(\$code) [*line 3736*]

Function Parameters:

- *int* **\$code** error code

translate an error code

- **Access** public

function patUser::updateStats() [*line 3309*]

update all statistics

- **Access** public

Class testHandler

[*line 5*]

class authHandler::testHandler

- **Package** patUser

function testHandler::patUserGetAuthData() [*line 13*]

patUserGetAuthData

this function is required.

function testHandler::patUserSetErrors(\$errors) [*line 50*]

Function Parameters:

- **\$errors**

patUserSetErrors

this function is optional

function testHandler::patUserSetRealm(\$realm) [*line 30*]

Function Parameters:

- **\$realm**

patUserRealm

This function is optional.

function testHandler::patUserSetUid(\$uid) [*line 40*]

Function Parameters:

- **\$uid**

patUserSetUid

this function is optional

Package patConfiguration Classes

Class patConfiguration

[line 11]

patConfiguration Class to read XML config files

- **Package** patConfiguration
- **Author** Stephan Schmidt < schst@php-tools.de>
- **Version** 1.4
- **Access** public

patConfiguration::\$cacheDir

string = "cache" [line 82]

directory where cache files are located

patConfiguration::\$conf

array = array() [line 34]

array that stores configuration

patConfiguration::\$currentConf

array = array() [line 40]

array that stores configuration from the current file

patConfiguration::\$data

```
string = "" [line 70]
```

current CDATA found

patConfiguration::\$defaultTypes

```
array = array() [line 100]
```

list of tags and the default types

patConfiguration::\$extensions

```
array = array() [line 46]
```

array that stores extensions

patConfiguration::\$externalFiles

```
array = array() [line 88]
```

list of all files that were needed

patConfiguration::\$includeDir

```
string = "" [line 76]
```

directory where include files are located

patConfiguration::\$nsStack

```
array = array() [line 52]
```

stack of the namespaces

patConfiguration::\$path

```
array = array() [line 28]
```

current path as array

patConfiguration::\$valDepth

```
int = 1 [line 64]
```

current depth of the stored values, i.e. array depth

patConfiguration::\$valStack

```
array = array() [line 58]
```

stack of values

patConfiguration::\$xmlFiles

```
array = array() [line 94]
```

all open files

patConfiguration::\$xmlSpecialchars

```
array = array(  
    "&" => "&",  
    "'" => "&apos;",  
    "\"" => "\"",  
    "<" => "<",  
    ">" => ">"  
)[line 17]
```

table used for translation of xml special chars

```
function patConfiguration::addExtension(&$ext, [$ns = ""]) [line 539]
```

Function Parameters:

- *object* `patConfigExtension` **&\$ext** extension that should be added
- *string* **\$ns** namespace for this extension (if differs from default ns)

add an extension

- **Access** public

```
function patConfiguration::appendData($data) [line 903]
```

Function Parameters:

- *mixed* **\$data** data to be appended

append Data to the current data

function patConfiguration::characterData(\$parser, \$data) *[line 841]*

Function Parameters:

- *int* **\$parser** resource id of the current parser
- *string* **\$data** character data, that was found

handle character data if the character data was found between tags using namespaces, the appropriate namespace handler will be called

function patConfiguration::clearConfigValue([\$path = ""]) *[line 1098]*

Function Parameters:

- *string* **\$path** path, where the value is stored

clears a config value if no path is given, the complete config will be cleared

- **Access public**

object function patConfiguration::createParser() *[line 1251]*

create a parser

function patConfiguration::endElement(\$parser, \$name) *[line 751]*

Function Parameters:

- *int* **\$parser** resource id of the current parser
- *string* **\$name** name of the element

handle end element if the end element contains a namespace calls the appropriate handler

function patConfiguration::externalEntity(\$parser, \$openEntityNames, \$base, \$systemId, \$publicId) *[line 1138]*

Function Parameters:

- **\$parser**
- **\$openEntityNames**
- **\$base**
- **\$systemId**
- **\$publicId**

mixed function patConfiguration::getConfigValue([\$path = ""]) [*line 1028*]

Function Parameters:

- *string* **\$path** path, where the value is stored

returns a configuration value if no path is given, all config values will be returned in an array

- **Access public**

function patConfiguration::loadCachedConfig(\$file, [\$mode = "w"]) [*line 154*]

Function Parameters:

- *string* **\$file** name of config file
- *string* **\$mode** mode of the parsing ("w" = overwrite old config, "a" = append to config)

load a configuration from a cache if cache is not valid, it will be updated automatically

- **Access public**

function patConfiguration::niceDie(\$method, \$message) [*line 1290*]

Function Parameters:

- *string* **\$method** method in which the error occurred
- *string* **\$message** the error message to display

generates a "nice" variant of die() with a few more interesting infos

function patConfiguration::parseConfigFile(\$file, [\$mode = "w"]) *[line 185]*

Function Parameters:

- *string* **\$file** name of the configuration file
- *string* **\$mode** mode of the parsing ("w" = overwrite old config, "a" = append to config)

parse a configuration file

- **Access** public

function patConfiguration::parseXMLFile(\$file) *[line 1217]*

Function Parameters:

- *string* **\$file** filename, without dirname

parse an external xml file

string function patConfiguration::replaceXMLSpecialchars(\$string, [\$table = array()]) *[line 1274]*

Function Parameters:

- *string* **\$string** string, where special chars should be replaced
- *array* **\$table** table used for replacing

replace XML special chars

function patConfiguration::setCacheDir(\$cacheDir) *[line 141]*

Function Parameters:

- *string* **\$cacheDir** name of the directory

set the directory, where all cache files are stored

- **Access** public

function patConfiguration::setConfigDir(\$configDir) [*line 119*]

Function Parameters:

- *string* **\$configDir** name of the directory

set the directory, where all xml config files are stored

- **Access** public

function patConfiguration::setConfigValue(\$path, \$value, [\$type = "leave"]) [*line 1070*]

Function Parameters:

- *string* **\$path** path, where the value will be stored
- *mixed* **\$value** value to store
- **\$type**

set a config value

*

- **Access** public

function patConfiguration::setConfigValues(\$values) [*line 1082*]

Function Parameters:

- *array* **\$values** assoc array containing paths and values

sets several config values

- **Access** public

function patConfiguration::setDefaultTypes(\$types, \$defaultTypes) [*line 108*]

Function Parameters:

- *array* **\$defaultTypes**
- **\$types**

set default types for tags

- **Access** public

function patConfiguration::setIncludeDir(\$includeDir) [*line 130*]

Function Parameters:

- *string* **\$includeDir** name of the directory

set the directory, where all extensions are stored

- **Access** public

function patConfiguration::startElement(\$parser, \$name, \$attributes) [*line 557*]

Function Parameters:

- *int* **\$parser** resource id of the current parser
- *string* **\$name** name of the element
- *array* **\$attributes** array containing all attributes of the element

handle start element if the start element contains a namespace calls the appropriate handler

function patConfiguration::writeConfigFile(\$filename, [\$format = "xml"], [\$options = array()]) [*line 294*]

Function Parameters:

- *string* **\$filename** name of the configfile
- *string* **\$format** format of the config file (xml or php)
- *array* **\$options** available options for php: varname => anyString ; available options for xml: mode => pretty

write a configfile format may be php or xml

- **Access** public

Package patTemplate Procedural Elements

patTemplate.php

- **Package** patTemplate

patTEMPLATE_TAG_END = "}" *[line 14]*

Variable suffix

- **Access** public

patTEMPLATE_TAG_START = "{" *[line 7]*

Variable prefix

- **Access** public

patTEMPLATE_TYPE_CONDITION = "CONDITION" *[line 30]*

Template type Condition

patTEMPLATE_TYPE_ODDEVEN = "ODDEVEN" *[line 25]*

Template type OddEven

patTEMPLATE_TYPE_SIMPLECONDITION = "SIMPLECONDITION" *[line 35]*

Template type SimpleCondition

patTEMPLATE_TYPE_STANDARD = "STANDARD" *[line 20]*

Template type Standard

Package patTemplate Classes

Class patTemplate

[line 48]

Easy-to-use but powerful template engine

Features include: several templates in one file, automatic repetitions, global variables, alternating lists, conditions, and much more

- **Package** patTemplate
- **Version** 2.5
- **Author** Stephan Schmidt < schst@php-tools.de>, Aaron Kalin <theiggsta@mapcore.com>, gERD Schaufelberger <gerd@php-tools.net>
- **Access** public

Constructor function patTemplate::patTemplate([\$type = "html"]) [line 61]

Function Parameters:

- *string* **\$type** type of output you want to generate.

Constructor

Create new patTemplate object You can choose between two outputs you want to generate: html (default) or tex (LaTeX). When "tex" is used the patTemplate markings used for variables are changed as LaTeX makes use of the default patTemplate markings. You can also change the markings later by calling setTags();

- **Access** public

function patTemplate::addGlobalVar(\$name, \$value) [*line 830*]

Function Parameters:

- *string* **\$name** name of the global variable
- *string* **\$value** value of the variable

Adds a global variable

Global variables are valid in all templates of this object

- See [patTemplate::addGlobalVars\(\)](#), [patTemplate::addVar\(\)](#), [patTemplate::addVars\(\)](#), [patTemplate::addRows\(\)](#)
- **Access** public

function patTemplate::addGlobalVars(\$variables, [\$prefix = ""]) [*line 845*]

Function Parameters:

- *array* **\$variables** array containing the variables
- *string* **\$prefix** prefix for variable names

Adds several global variables

Global variables are valid in all templates of this object \$variables is an associative array, containing name/value pairs of the variables

- See [patTemplate::addGlobalVar\(\)](#), [patTemplate::addVar\(\)](#), [patTemplate::addVars\(\)](#), [patTemplate::addRows\(\)](#)
- **Access** public

function patTemplate::addRows(\$template, \$rows, [\$prefix = ""]) *[line 793]*

Function Parameters:

- *string* **\$template** name of the template
- *array* **\$rows** array containing associative arrays with variable/value pairs
- *string* **\$prefix** prefix for all variable names

Adds several rows of variables to a template

Each Template can have an unlimited amount of its own variables. Can be used to add a database result as variables to a template.

- See [patTemplate::addVar\(\)](#), [patTemplate::addVars\(\)](#), [patTemplate::addGlobalVar\(\)](#), [patTemplate::addGlobalVars\(\)](#)
- **Access** public

function patTemplate::addTemplate(\$name, \$filename) *[line 174]*

Function Parameters:

- *string* **\$name** name of the template
- *string* **\$filename** filename of the sourcetemplate

Add a template

Adds a plain text/html to the template engine. The file has to be in the directory that has been set using `setBaseDir`.

- See [setBaseDir\(\)](#), [patTemplate::addTemplates\(\)](#)
- **Deprecated** 2.4 2001/11/05
- **Access** public

function patTemplate::addTemplates(\$templates) *[line 192]*

Function Parameters:

- *array* **\$templates** associative Array with name/filename pairs

Adds several templates

Adds several templates to the template engine using an associative array. Names of the templates are stored in the keys, filenames are the values. The templates have to be in the directory set by `setBaseDir()`.

- **See** `setBaseDir()`, [patTemplate::addTemplate\(\)](#)
- **Deprecated** 2.4 2001/11/05
- **Access** public

```
function patTemplate::addVar($template, $name, $value) [line 735]
```

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$name** name of the variables
- *mixed* **\$value** value of the variable

Adds a variable to a template

Each Template can have an unlimited amount of its own variables

- **See** [patTemplate::addVars\(\)](#), [patTemplate::addRows\(\)](#), [patTemplate::addGlobalVar\(\)](#), [patTemplate::addGlobalVars\(\)](#)
- **Access** public

```
function patTemplate::addVars($template, $variables, [$prefix = ""]) [line 767]
```

Function Parameters:

- *string* **\$template** name of the template
- *array* **\$variables** associative array of the variables
- *string* **\$prefix** prefix for all variable names

Adds several variables to a template

Each Template can have an unlimited amount of its own variables. \$variables has to be an associative array containing variable/value pairs

- See [patTemplate::addVar\(\)](#), [patTemplate::addRows\(\)](#), [patTemplate::addGlobalVar\(\)](#), [patTemplate::addGlobalVars\(\)](#)
- Access public

function patTemplate::clearAllTemplates() [*line 1332*]

clears all templates

- Access public

function patTemplate::clearAttribute(\$template, \$attribute) [*line 337*]

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$attribute** name of the attribute

Clears an attribute of a template

supported attributes: visibilty, loop, parse, unusedvars

- See [patTemplate::setAttribute\(\)](#), [patTemplate::setAttributes\(\)](#), [patTemplate::getAttribute\(\)](#)
- Access public

function patTemplate::clearTemplate(\$name) [*line 1320*]

Function Parameters:

- *string* **\$name** name of the template

clears a parsed Template

parsed Content, variables and the loop attribute are cleared

- **Access** public

function patTemplate::displayParsedTemplate([\$name = ""]) [*line 1235*]

Function Parameters:

- *string* **\$name** name of the template

displays a parsed Template

If the template has not been loaded, it will be loaded.

- **See** [patTemplate::getParsedTemplate\(\)](#)
- **Access** public

function patTemplate::dump(\$name) [*line 1467*]

Function Parameters:

- *string* **\$name** name of template - repetitive argument

displays useful information about all or named templates

returns content, variables, attributes and unused variables

- **Access** public

bool function patTemplate::exists(\$name) [*line 151*]

Function Parameters:

- *string* **\$name** name of the template

Check if a template exists

- **Access** public

mixed function patTemplate::getAttribute(\$template, \$attribute) [*line 319*]

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$attribute** name of the attribute

Gets an attribute of a template

supported attributes: visibility, loop, parse, unusedvars

- **See** [patTemplate::setAttribute\(\)](#), [patTemplate::setAttributes\(\)](#), [patTemplate::clearAttribute\(\)](#)
- **Access** public

string function patTemplate::getParsedTemplate([\$name = ""]) [*line 1182*]

Function Parameters:

- *string* **\$name** name of the template

returns a parsed Template

If the template already has been parsed, it just returns the parsed template. If the template has not been loaded, it will be loaded.

- See [patTemplate::displayParsedTemplate\(\)](#)
- Access public

mixed function patTemplate::getVar(\$template, \$var, \$index) *[line 1427]*

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$var** name of the variable
- *integer* **\$index** no of repetition

get the value of a variable

function patTemplate::parseTemplate(\$template, [\$mode = "w"]) *[line 931]*

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$mode** mode for the parsing

parses a template

Parses a template and stores the parsed content. mode can be "w" for write (delete already parsed content) or "a" for append (appends the new parsed content to the already parsed content)

- See [parseStandardTemplate\(\)](#), [parseIterativeTemplate\(\)](#)
- Access public

function patTemplate::readTemplatesFromFile(\$file) [*line 372*]

Function Parameters:

- *string* **\$file** filename

Parses several templates from one patTemplate file

Templates can be separated using Tags. The file has to be located in the directory that has been set using `setBaseDir`.

- See [patTemplate::setBasedir\(\)](#)
- **Access** public

function patTemplate::setAttribute(\$template, \$attribute, \$value) [*line 277*]

Function Parameters:

- *string* **\$template** name of the template
- *string* **\$attribute** name of the attribute
- *mixed* **\$value** value of the attribute

Sets an attribute of a template

supported attributes: visibility, loop, parse, unusedvars

- See [patTemplate::setAttributes\(\)](#), [patTemplate::getAttribute\(\)](#), [patTemplate::clearAttribute\(\)](#)
- **Access** public

function patTemplate::setAttributes(\$template, \$attributes) [*line 294*]

Function Parameters:

- *string* **\$template** name of the template
- *array* **\$attributes** attribute/value pairs

Sets several attribute of a template

\$attributes has to be a associative arrays containing attribute/value pairs supported attributes: visibilty, loop, parse, unusedvars

- See [patTemplate::setAttribute\(\)](#), [patTemplate::getAttribute\(\)](#), [patTemplate::clearAttribute\(\)](#)
- Access public

function patTemplate::setBasedir(\$basedir) [*line 140*]

Function Parameters:

- *string* **\$basedir** directory of the templates

Set template directory

Sets the directory where the template are stored. By default the engine looks in the directory where the original file is stored.

- Access public

function patTemplate::setTags([\$start = patTEMPLATE_TAG_START], [\$end = patTEMPLATE_TAG_END]) [*line 125*]

Function Parameters:

- *string* **\$start** start tag
- *string* **\$end** end tag

Set template tags

Sets the start and end tags of template variables

- **Access** public

function patTemplate::setType([\$type = ""]) [*line 104*]

Function Parameters:

- *string* **\$type** predefined template type, like "html" or "tex"

Set template type

select a predefined template type

- **Access** public

Appendices

Appendix A - Class Trees

Package patConfiguration

patConfiguration

- [patConfiguration](#)

Package patUser

myClass

- [myClass](#)

patUser

- [patUser](#)

testHandler

- [testHandler](#)

Package patTemplate

patTemplate

- [patTemplate](#)

Appendix D - Todo List

In Package patUser

In [patUser.php](#)

- check user

Index

C

constructor patTemplate::patTemplate()	63
<i>Constructor</i>	
constructor patUser::patUser()	20
<i>create new user object</i>	
constructor patUser::__construct()	20
<i>create new user object</i>	

E

example modifyUser.php	7
example patConfiguration.php	8
example getPermissions.php	6
example cryptFunction.php	5
<i>example_cryptFunction</i>	
example2.php	3
example authHandler.php	4
example.php	2

M

myClass::crypter()	13
<i>example crypt inside objects function</i>	
myClass	13
<i>demo class for external crypt functions</i>	
myCrypter()	5
<i>demo crypt function</i>	

P

patUser::storeSessionValue()	48
<i>store a value in the session</i>	
patUser::setUnauthorizedURL()	48
<i>set URL to redirect unauthorized users to</i>	
patUser::translateErrorCode()	49
<i>translate an error code</i>	
patUser::updateStats()	49
<i>update all statistics</i>	
patConfiguration	51
<i>patConfiguration</i>	
<i>Class to read XML config files</i>	
patUser::setUnauthorizedTemplate()	47

	<i>Sets the name of the template to use for unauthorized access, to override the default 'patUserUnauthorized.tpl'</i>	
patUser::setTemplate()	<i>set template object the template object is used for displaying login / logout screens</i>	47
patUser::setMaxLoginAttempts()	<i>set maximum amount of login attempts</i>	45
patUser::setLoginTemplate()	<i>Sets the name of the login template to use, to override the default 'patUserLogin.tpl'</i>	45
patUser::setPermFields()	<i>set fieldnames that are used in the permissions</i>	46
patUser::setPermTable()	<i>set tablename that contains permissions</i>	46
patUser::setRealm()	<i>set authentication realm</i>	47
patConfiguration::\$cacheDir	<i>directory where cache files are located</i>	51
patConfiguration::\$conf	<i>array that stores configuration</i>	51
patConfiguration::\$path	<i>current path as array</i>	52
patConfiguration::\$nsStack	<i>stack of the namespaces</i>	52
patConfiguration::\$valDepth	<i>current depth of the stored values, i.e. array depth</i>	52
patConfiguration::\$valStack	<i>stack of values</i>	53
patConfiguration::\$xmlFiles	<i>all open files</i>	53
patConfiguration::\$includeDir	<i>directory where include files are located</i>	52
patConfiguration::\$externalFiles	<i>list of all files that were needed</i>	52
patConfiguration::\$currentConf	<i>array that stores configuration from the current file</i>	51
patConfiguration::\$data	<i>current CDATA found</i>	52
patConfiguration::\$defaultTypes	<i>list of tags and the default types</i>	52
patConfiguration::\$extensions	<i>array that stores extensions</i>	52
patUser::setGroupTable()	<i>set tablename that contains group info</i>	45
patUser::setGroupRelTable()	<i>set tablename that contains user group relations</i>	44
patUser::logOut()	<i>force log out of an authenticated user</i>	37
patUser::keepHistory()	<i>let patUser create a history of visited pages</i>	37
patUser::modifyGroup()	<i>modify an existing group</i>	38
patUser::modifyUser()		38

<i>modify an existing user</i>	
patUser::removeLastError()	39
<i>remove the last error from the list</i>	
patUser::issetSessionValue()	37
<i>check, whether a value has been stored in the session</i>	
<i>may only be used if patUser was called with \$useSessions = true</i>	
patUser::isMemberOfGroup()	36
<i>check, if a user is in a group</i>	
patUser::hasPermission()	35
<i>check, whether user has a permission</i>	
patUser::identifyGroup()	35
<i>identify a group by certain fields</i>	
patUser::identifyUser()	36
<i>identify a user by certain fields</i>	
patUser::isAuthenticated()	36
<i>check, whether a user is already authenticated</i>	
patUser::removeUserFromGroup()	39
<i>delete user(s) from group(s)</i>	
patUser::requireAuthentication()	39
<i>require an authenticated user</i>	
patUser::setCryptFunction()	43
<i>set function that is used for passwd encryption</i>	
patUser::setAuthTable()	42
<i>set tablename that contains auth info</i>	
patUser::setError()	43
<i>set error code</i>	
patUser::setGroupFields()	43
<i>set fieldnames that contain group info</i>	
patUser::setGroupRelFields()	44
<i>set fieldnames that are used in the user group relations</i>	
patUser::setAuthHandler()	42
<i>set authentication handler object</i>	
patUser::setAuthFields()	42
<i>set fieldnames that contain auth info</i>	
patUser::searchUsers()	40
<i>Search for users matching certain criterias</i>	
patUser::sendQuery()	40
<i>send query to any database</i>	
<i>can be used the use the internal dbcs in your own apps without knowing how the user object was configured</i>	
patUser::setAuthDbc()	41
<i>set dbc that contains auth info</i>	
patUser::setAuthDsn()	41
<i>set dsn that contains authentication information</i>	
patConfiguration::\$xmlSpecialchars	53
<i>table used for translation of xml special chars</i>	
patConfiguration::addExtension()	53
<i>add an extension</i>	
patTemplate::addVars()	66
<i>Adds several variables to a template</i>	
patTemplate::addVar()	66
<i>Adds a variable to a template</i>	
patTemplate::clearAllTemplates()	67

<i>clears all templates</i>	
patTemplate::clearAttribute()	67
<i>Clears an attribute of a template</i>	
patTemplate::clearTemplate()	67
<i>clears a parsed Template</i>	
patTemplate::addTemplates()	65
<i>Adds several templates</i>	
patTemplate::addTemplate()	65
<i>Add a template</i>	
patTemplate	63
<i>Easy-to-use but powerful template engine</i>	
patTEMPLATE_TYPE_STANDARD	62
<i>Template type Standard</i>	
patTemplate::addGlobalVar()	64
<i>Adds a global variable</i>	
patTemplate::addGlobalVars()	64
<i>Adds several global variables</i>	
patTemplate::addRows()	65
<i>Adds several rows of variables to a template</i>	
patTemplate::displayParsedTemplate()	68
<i>displays a parsed Template</i>	
patTemplate::dump()	68
<i>displays useful information about all or named templates</i>	
patTemplate::setAttributes()	71
<i>Sets several attribute of a template</i>	
patTemplate::setAttribute()	71
<i>Sets an attribute of a template</i>	
patTemplate::setBasedir()	72
<i>Set template directory</i>	
patTemplate::setTags()	72
<i>Set template tags</i>	
patTemplate::setType()	73
<i>Set template type</i>	
patTemplate::readTemplatesFromFile()	71
<i>Parses several templates from one patTemplate file</i>	
patTemplate::parseTemplate()	70
<i>parses a template</i>	
patTemplate::exists()	69
<i>Check if a template exists</i>	
patTemplate::getAttribute()	69
<i>Gets an attribute of a template</i>	
patTemplate::getParsedTemplate()	69
<i>returns a parsed Template</i>	
patTemplate::getVar()	70
<i>get the value of a variable</i>	
patTEMPLATE_TYPE_SIMPLECONDITION	61
<i>Template type SimpleCondition</i>	
patTEMPLATE_TYPE_ODDEVEN	61
<i>Template type OddEven</i>	
patConfiguration::loadCachedConfig()	55
<i>load a configuration from a cache</i>	
<i>if cache is not valid, it will be updated automatically</i>	
patConfiguration::getConfigValue()	55

<i>returns a configuration value</i>	
<i>if no path is given, all config values will be returned in an array</i>	
patConfiguration::niceDie()	55
<i>generates a "nice" variant of die() with a few more interesting infos</i>	
patConfiguration::parseConfigFile()	56
<i>parse a configuration file</i>	
patConfiguration::parseXMLFile()	56
<i>parse an external xml file</i>	
patConfiguration::externalEntity()	54
patConfiguration::endElement()	54
<i>handle end element</i>	
<i>if the end element contains a namespace calls the appropriate handler</i>	
patConfiguration::appendData()	53
<i>append Data to the current data</i>	
patConfiguration::characterData()	54
<i>handle character data</i>	
<i>if the character data was found between tags using namespaces, the appropriate namespace handler will be called</i>	
patConfiguration::clearConfigValue()	54
<i>clears a config value</i>	
<i>if no path is given, the complete config will be cleared</i>	
patConfiguration::createParser()	54
<i>create a parser</i>	
patConfiguration::replaceXMLSpecialchars()	56
<i>replace XML special chars</i>	
patConfiguration::setCacheDir()	56
<i>set the directory, where all cache files are stored</i>	
patTemplate.php	61
patConfiguration::writeConfigFile()	59
<i>write a configfile</i>	
<i>format may be php or xml</i>	
patTEMPLATE_TAG_END	61
<i>Variable suffix</i>	
patTEMPLATE_TAG_START	61
<i>Variable prefix</i>	
patTEMPLATE_TYPE_CONDITION	61
<i>Template type Condition</i>	
patConfiguration::startElement()	59
<i>handle start element</i>	
<i>if the start element contains a namespace calls the appropriate handler</i>	
patConfiguration::setIncludeDir()	58
<i>set the directory, where all extensions are stored</i>	
patConfiguration::setConfigDir()	57
<i>set the directory, where all xml config files are stored</i>	
patConfiguration::setConfigValue()	57
<i>set a config value</i>	
patConfiguration::setConfigValues()	58
<i>sets several config values</i>	
patConfiguration::setDefaultTypes()	58
<i>set default types for tags</i>	
patUser::goHistory()	35
<i>go back x pages in the history</i>	
<i>as Header("Location:...") is used, there must not be any output before this is</i>	

<i>called</i>	34
patUser::getUsersInGroup()	34
<i>get users in group</i>	
patUser::\$cryptSalt	15
<i>salt string for stronger encryption</i>	
patUser::\$cryptFunction	15
<i>name of function, that should be used for passwd encryption</i>	
patUser::\$dbcs	15
<i>all dbcs</i>	
patUser::\$errorMessages	16
<i>error messages</i>	
patUser::\$errors	16
<i>all codes of the errors that happened while processing</i>	
patUser::\$authTable	15
<i>Table that stores the authentication data</i>	
patUser::\$authHandler	14
<i>authentication handler object (false if no handler is used)</i>	
patUser	14
<i>user management class</i>	
prepend.php	12
patUser::\$actionVar	14
<i>name of the global variable that indicates the action in the request</i>	
patUser::\$authenticated	14
<i>state of user/session: authenticated or not</i>	
patUser::\$authFields	14
<i>Fieldnames in the authentication table</i>	
patUser::\$fieldLocs	16
<i>locations (table/fieldname) of fields</i>	
patUser::\$groupFields	16
<i>Fieldnames in the group table</i>	
patUser::\$permTable	17
<i>Table that stores the permissions</i>	
patUser::\$permsConv	17
<i>table to convert permissions</i>	
patUser::\$realm	18
<i>default realm for HTTP authentication</i>	
patUser::\$relFields	18
<i>Fieldnames of the user-group relation table</i>	
patUser::\$relTable	18
<i>Table that stores the user - group relations</i>	
patUser::\$perms	17
<i>Possible permissions</i>	
patUser::\$permFields	17
<i>Fieldnames in the permissions table</i>	
patUser::\$groupTable	16
<i>Table that stores the group data</i>	
patUser::\$ignoreVars	17
<i>variable names that should be ignored by getSelfUrl()</i>	
patUser::\$loginTemplate	17
<i>filename of the template used for login</i>	
patUser::\$maxLoginAttempts	17
<i>maximum login attempts for a session</i>	
patUSER_USER_ALREADY_EXISTS	11

<i>error code: user already exists</i>	
<u>patUSER TABLE DOES NOT EXIST</u>	11
<i>error code: table does not exist</i>	
<u>patUSER NEED GROUPNAME</u>	9
<i>error code: function requires a group name</i>	
<u>patUSER NEED GID</u>	9
<i>error code: function requires group id</i>	
<u>patUSER NEED ID</u>	10
<i>error code: function requires user or group id</i>	
<u>patUSER NEED ID TYPE</u>	10
<i>error code: function requires type of supplied id (user or group)</i>	
<u>patUSER NEED PASSWD</u>	10
<i>error code: function requires a password</i>	
<u>patUSER LOGIN DISABLED</u>	9
<i>error code: login for this user was disabled</i>	
<u>patUSER INSERT NOT ALLOWED</u>	9
<i>error code: insert is not allowed</i>	
<u>patUSER ALREADY JOINED GROUP</u>	9
<i>error code: user already is in group</i>	
<u>patUSER COULD NOT IDENTIFY LINE</u>	9
<i>error code: could not identify line in table</i>	
<u>patUSER DELETE NOT ALLOWED</u>	9
<i>error code: delete is not allowed</i>	
<u>patUSER GROUP ALREADY EXISTS</u>	9
<i>error code: group already exists (when adding a group)</i>	
<u>patUSER NEED UID</u>	10
<i>error code: function requires a user id</i>	
<u>patUSER NEED USERNAME</u>	10
<i>error code: function requires a user name</i>	
<u>patUSER NO UNIQUE PRIMARY FOUND</u>	10
<i>error code: data matched more than one row</i>	
<u>patUSER NO UNIQUE GROUP FOUND</u>	10
<i>error code: no unique group matched the query</i>	
<u>patUSER NO UNIQUE USER FOUND</u>	11
<i>error code: more than one user matched the query</i>	
<u>patUSER NO USER FOUND</u>	11
<i>error code: no user matched the query</i>	
<u>patUSER PASSWD MISMATCH</u>	11
<i>error code: password incorrect</i>	
<u>patUSER NO PRIMARY FOUND</u>	10
<i>error code: no primary key was found</i>	
<u>patUSER NO GROUP FOUND</u>	10
<i>error code: no group matched the query</i>	
<u>patUSER NOT IN GROUP</u>	10
<i>error code: user is not in group</i>	
<u>patUSER NO DATA CHANGED</u>	10
<i>error code: no data was changed (affected rows = 0)</i>	
<u>patUSER NO DATA GIVEN</u>	10
<i>error code: function requires data</i>	
<u>patUSER NO DB RESULT</u>	10
<i>error code: query had no result</i>	
<u>patUser::\$sessionVar</u>	18
<i>name of the global variable used for sessions</i>	

patUser::\$stats	18
<i>all statistic options</i>	
patUser::getGroupData()	29
<i>fetch the group data from group table</i>	
patUser::getField()	28
<i>fetch the value of a field</i>	
patUser::getGroupFields()	29
<i>get fieldnames that contain group info</i>	
patUser::getGroups()	29
<i>get groups from the grouptable</i>	
patUser::getHistory()	30
<i>get the history for the current user</i>	
patUser::getDbc()	28
<i>get the dbc object for a table</i>	
patUser::getAuthFields()	28
<i>get fieldnames that contain auth info</i>	
patUser::encryptPasswd()	27
<i>encrypt a password using the specified encryption function</i>	
patUser::deleteUser()	27
<i>delete (data of) an existing user</i>	
patUser::getAllErrorCodes()	27
<i>get all error codes</i>	
patUser::getAllErrorMessages()	28
<i>get all error messages</i>	
patUser::getAllErrors()	28
<i>get all error codes AND messages</i>	
patUser::getJoinedGroups()	30
<i>get joined groups</i>	
patUser::getLastError()	30
<i>get the last error code AND message</i>	
patUser::getTableInfo()	33
<i>get information about table fields like DESCRIBE [table]</i>	
patUser::getTableDef()	33
<i>get table definitions</i>	
patUser::getUid()	33
<i>an easy way to recieve the id if the current user.</i>	
patUser::getUserData()	33
<i>fetch the user data from a table</i>	
patUser::getUsers()	34
<i>get users from any table</i>	
patUser::getSessionValue()	32
<i>get a value that was stored in the session</i>	
patUser::getPrimaryValue()	32
<i>get the value of the primary key in an additional table identifies a unique line in an additional table</i>	
patUser::getLastErrorCode()	31
<i>get the last error code</i>	
patUser::getLastErrorMessage()	31
<i>get the last error message</i>	
patUser::getPermissions()	31
<i>get all permissions of a user or all permissions</i>	
patUser::getPermTable()	31

<i>get tablename that contains permissions</i>	26
patUser::deletePermission()	26
<i>delete permission(s)</i>	
patUser::deleteGroup()	26
<i>delete an existing group</i>	
patUser::\$userIdSequence	19
<i>name of the sequence for user ids</i>	
patUser::\$usePermissions	19
<i>flag to indicate whether permsnsions are available</i>	
patUser::\$useSessions	20
<i>flag to indicate whether sessions are used</i>	
patUser::\$useTemplate	20
<i>flag to indicate, whether template is used for output</i>	
patUser::addDbc()	21
<i>add a dsn or dbc-object</i>	
patUser::\$unauthorizedURL	19
<i>URL to redirect unauthorized users to</i>	
patUser::\$unauthorizedTemplate	19
<i>filename of the template used for unauthorized users</i>	
patUser::\$statsUpdated	18
<i>whether user stats have been updated.</i>	
patUser::\$systemVars	18
<i>information about the project</i>	
patUser::\$tables	19
<i>all tables that are used</i>	
patUser::\$tpl	19
<i>variable to store the patTemplate object (false if no template is used)</i>	
patUser::addFieldLocation()	21
<i>set the location of a field</i>	
patUser::addGroup()	21
<i>adds a group</i>	
patUser::clearSessionValue()	25
<i>clear a value in the session</i>	
patUser::clearErrors()	24
<i>clear all errors</i>	
patUser::convertOptionsToSql()	25
<i>convert options to sql</i>	
patUser::countTableEntries()	25
<i>count amount of entries for a user in a table</i>	
patUser::deleteAllPermissions()	26
<i>delete all permissions without setting new ones</i>	
patUser::authenticate()	24
<i>authenticate a user</i>	
patUser::addUserToGroup()	24
<i>add a user to a group</i>	
patUser::addPermission()	22
<i>add permission(s)</i>	
patUser::addStats()	22
<i>add a statistic option</i>	
patUser::addTable()	23
<i>add table that is used to store user data</i>	
patUser::addUser()	23
<i>adds a user</i>	

patUser.php	9
<i>Powerfull user/group/permission management class based on databases.</i>	

T

testHandler::patUserSetUid()	50
<i>patUserSetUid</i>	
testHandler::patUserSetRealm()	50
<i>patUserRealm</i>	
testHandler::patUserSetErrors()	50
<i>patUserSetErrors</i>	
testHandler::patUserGetAuthData()	49
<i>patUserGetAuthData</i>	
testHandler	49
<i>class authHandler::testHandler</i>	