

# patServer



# Contents

<a href="#">Package patServer Procedural Elements</a>	2
<a href="#">flashchat.php</a>	2
<a href="#">httpd.php</a>	3
<a href="#">jabber.php</a>	4
<a href="#">patTTT.php</a>	5
<a href="#">telnetchat.php</a>	6
<a href="#">telnetchat_pear.php</a>	7
<a href="#">Package patServer Classes</a>	8
<a href="#">Class Net_Server_TelnetChat</a>	8
<a href="#">Var \$users</a>	8
<a href="#">Method getClientIdByNick</a>	8
<a href="#">Class patChatServer</a>	9
<a href="#">Method onClose</a>	9
<a href="#">Method onConnect</a>	9
<a href="#">Method onConnectionRefused</a>	9
<a href="#">Method onReceiveData</a>	9
<a href="#">Method onShutdown</a>	10
<a href="#">Method onStart</a>	10
<a href="#">Class patFlashChatServer</a>	10
<a href="#">Var \$dbc</a>	10
<a href="#">Var \$readEndCharacter</a>	10
<a href="#">Var \$user</a>	10
<a href="#">Var \$users</a>	11
<a href="#">Method doLogin</a>	11
<a href="#">Method onClose</a>	11
<a href="#">Method onReceiveRequest</a>	11
<a href="#">Method processTextMessage</a>	12
<a href="#">Method quit</a>	12
<a href="#">Class patHTTPServer</a>	12
<a href="#">Var \$readEndCharacter</a>	12
<a href="#">Method getContentType</a>	13
<a href="#">Method onReceiveData</a>	13
<a href="#">Method onStart</a>	13
<a href="#">Method parsePath</a>	13
<a href="#">Method parseRequest</a>	14
<a href="#">Method stripTrailingSlash</a>	14
<a href="#">Class patJabberServer</a>	14
<a href="#">Var \$attStack</a>	15
<a href="#">Var \$inMessage</a>	15
<a href="#">Var \$inStream</a>	15
<a href="#">Var \$readBufferSize</a>	15
<a href="#">Var \$tagStack</a>	15
<a href="#">Var \$xmlParser</a>	15

<a href="#">Method onClose</a>	16
<a href="#">Method cData</a>	16
<a href="#">Method destroyParser</a>	16
<a href="#">Method endElement</a>	16
<a href="#">Method startElement</a>	16
<a href="#">Class patPaintServer</a>	16
<a href="#">Method onReceiveData</a>	17
<a href="#">Class patServer</a>	17
<a href="#">Var \$clientFD</a>	17
<a href="#">Var \$clientInfo</a>	18
<a href="#">Var \$clients</a>	18
<a href="#">Var \$debug</a>	18
<a href="#">Var \$debugDest</a>	18
<a href="#">Var \$debugMode</a>	18
<a href="#">Var \$domain</a>	18
<a href="#">Var \$maxClients</a>	18
<a href="#">Var \$maxQueue</a>	18
<a href="#">Var \$null</a>	19
<a href="#">Var \$port</a>	19
<a href="#">Var \$readBufferSize</a>	19
<a href="#">Var \$readEndCharacter</a>	19
<a href="#">Constructor patServer</a>	19
<a href="#">Method acceptConnection</a>	19
<a href="#">Method broadcastData</a>	20
<a href="#">Method closeConnection</a>	20
<a href="#">Method getClientInfo</a>	21
<a href="#">Method getClients</a>	21
<a href="#">Method getLastSocketError</a>	21
<a href="#">Method isConnected</a>	21
<a href="#">Method sendData</a>	22
<a href="#">Method setDebugMode</a>	22
<a href="#">Method setMaxClients</a>	23
<a href="#">Method shutDown</a>	23
<a href="#">Method start</a>	23
<a href="#">Class patTelnetChatServer</a>	24
<a href="#">Var \$users</a>	24
<a href="#">Method getClientIdByNick</a>	24
<a href="#">Class patTicTacToe</a>	24
<a href="#">Var \$candidates</a>	25
<a href="#">Var \$emptyField</a>	25
<a href="#">Var \$gameId</a>	25
<a href="#">Var \$games</a>	25
<a href="#">Var \$players</a>	25
<a href="#">Var \$readEndCharacter</a>	26
<a href="#">Var \$waitingPlayers</a>	26
<a href="#">Method checkForNewGames</a>	26
<a href="#">Method createNewGame</a>	26
<a href="#">Method endGame</a>	26
<a href="#">Method getGameId</a>	26

<a href="#">Method getOpponentId</a>	26
<a href="#">Method onClose</a>	27
<a href="#">Method onReceiveRequest</a>	27
<a href="#">Method putOnHold</a>	27
<a href="#">Method sendResponseToPlayers</a>	27
<a href="#">Method startGame</a>	28
<a href="#">Class patXMLChatServer</a>	28
<a href="#">Var \$users</a>	28
<a href="#">Method doLogin</a>	28
<a href="#">Method onClose</a>	28
<a href="#">Method onReceiveData</a>	29
<a href="#">Class patXMLServer</a>	29
<a href="#">Var \$parser</a>	29
<a href="#">Method buildXML</a>	29
<a href="#">Method characterData</a>	29
<a href="#">Method endElement</a>	30
<a href="#">Method onStart</a>	30
<a href="#">Method parseXML</a>	30
<a href="#">Method startElement</a>	30
<a href="#">Class patXMLServer_Dom</a>	30
<a href="#">Method broadcastResponse</a>	31
<a href="#">Method encodeResponse</a>	31
<a href="#">Method sendResponse</a>	31
<a href="#">Package patDbc Procedural Elements</a>	34
<a href="#">patDbc.php</a>	34
<a href="#">Define patDBC_CLOSED</a>	34
<a href="#">Define patDBC_OPEN</a>	34
<a href="#">Define patDBC_TYPEASSOC</a>	34
<a href="#">Define patDBC_TYPEBOTH</a>	34
<a href="#">Define patDBC_TYPENUM</a>	34
<a href="#">Package patDbc Classes</a>	35
<a href="#">Class patDbcResult</a>	35
<a href="#">Var \$id</a>	35
<a href="#">Method dump</a>	35
<a href="#">Method dumpRow</a>	36
<a href="#">Method fetch_array</a>	36
<a href="#">Method fetch_row</a>	36
<a href="#">Method free</a>	36
<a href="#">Method get_result</a>	37
<a href="#">Method num_fields</a>	37
<a href="#">Method num_rows</a>	37
<a href="#">Method setIdentifier</a>	37
<a href="#">Class patMySqlDbc</a>	38
<a href="#">Var \$dbhost</a>	38
<a href="#">Var \$dbid</a>	38
<a href="#">Var \$dbname</a>	38
<a href="#">Var \$dbpass</a>	38
<a href="#">Var \$dbuser</a>	38

<a href="#">Var \$status</a>	38
<a href="#">Constructor patMySqlDbc</a>	38
<a href="#">Method affected_rows</a>	39
<a href="#">Method close</a>	39
<a href="#">Method connect</a>	39
<a href="#">Method create_db</a>	39
<a href="#">Method drop_db</a>	40
<a href="#">Method get_fields</a>	40
<a href="#">Method insert_id</a>	41
<a href="#">Method query</a>	41
<a href="#">Class patMySqlResult</a>	41
<a href="#">Method data_seek</a>	41
<a href="#">Method fetch_array</a>	42
<a href="#">Method fetch_field</a>	42
<a href="#">Method fetch_object</a>	43
<a href="#">Method fetch_row</a>	43
<a href="#">Method field_len</a>	43
<a href="#">Method field_name</a>	43
<a href="#">Method field_seek</a>	44
<a href="#">Method free</a>	44
<a href="#">Method num_fields</a>	44
<a href="#">Method num_rows</a>	45

## [Package patUser Procedural Elements](#) 47

<a href="#">patUser.php</a>	47
<a href="#">Define patUSER_ALREADY_JOINED_GROUP</a>	47
<a href="#">Define patUSER_COULD_NOT_IDENTIFY_LINE</a>	47
<a href="#">Define patUSER_DELETE_NOT_ALLOWED</a>	47
<a href="#">Define patUSER_GROUP_ALREADY_EXISTS</a>	48
<a href="#">Define patUSER_INSERT_NOT_ALLOWED</a>	48
<a href="#">Define patUSER_LOGIN_DISABLED</a>	48
<a href="#">Define patUSER_NEED_GID</a>	48
<a href="#">Define patUSER_NEED_GROUPNAME</a>	48
<a href="#">Define patUSER_NEED_ID</a>	49
<a href="#">Define patUSER_NEED_ID_TYPE</a>	49
<a href="#">Define patUSER_NEED_PASSWD</a>	49
<a href="#">Define patUSER_NEED_UID</a>	49
<a href="#">Define patUSER_NEED_USERNAME</a>	49
<a href="#">Define patUSER_NOT_IN_GROUP</a>	50
<a href="#">Define patUSER_NO_DATA_CHANGED</a>	50
<a href="#">Define patUSER_NO_DATA_GIVEN</a>	50
<a href="#">Define patUSER_NO_DB_RESULT</a>	50
<a href="#">Define patUSER_NO_GROUP_FOUND</a>	51
<a href="#">Define patUSER_NO_PRIMARY_FOUND</a>	51
<a href="#">Define patUSER_NO_UNIQUE_GROUP_FOUND</a>	51
<a href="#">Define patUSER_NO_UNIQUE_PRIMARY_FOUND</a>	51
<a href="#">Define patUSER_NO_UNIQUE_USER_FOUND</a>	51
<a href="#">Define patUSER_NO_USER_FOUND</a>	52
<a href="#">Define patUSER_PASSWD_MISMATCH</a>	52
<a href="#">Define patUSER_TABLE_DOES_NOT_EXIST</a>	52

<a href="#">Define patUSER USER ALREADY EXISTS</a>	52
<a href="#">Package patUser Classes</a>	53
<a href="#">Class patUser</a>	53
<a href="#">  Var \$actionVar</a>	53
<a href="#">  Var \$authenticated</a>	53
<a href="#">  Var \$authFields</a>	53
<a href="#">  Var \$authTable</a>	54
<a href="#">  Var \$cryptFunction</a>	54
<a href="#">  Var \$dbcs</a>	54
<a href="#">  Var \$errorMessages</a>	54
<a href="#">  Var \$errors</a>	55
<a href="#">  Var \$fieldLocs</a>	55
<a href="#">  Var \$groupFields</a>	55
<a href="#">  Var \$groupTable</a>	55
<a href="#">  Var \$ignoreVars</a>	55
<a href="#">  Var \$loginTemplate</a>	55
<a href="#">  Var \$maxLoginAttempts</a>	55
<a href="#">  Var \$options</a>	55
<a href="#">  Var \$outputErrors</a>	56
<a href="#">  Var \$permFields</a>	56
<a href="#">  Var \$perms</a>	56
<a href="#">  Var \$permsConv</a>	56
<a href="#">  Var \$permTable</a>	56
<a href="#">  Var \$realm</a>	56
<a href="#">  Var \$relFields</a>	56
<a href="#">  Var \$relTable</a>	57
<a href="#">  Var \$sessionVar</a>	57
<a href="#">  Var \$stats</a>	57
<a href="#">  Var \$systemVars</a>	57
<a href="#">  Var \$tables</a>	57
<a href="#">  Var \$tpl</a>	57
<a href="#">  Var \$unauthorizedTemplate</a>	57
<a href="#">  Var \$unauthorizedURL</a>	58
<a href="#">  Var \$useSessions</a>	58
<a href="#">  Var \$useTemplate</a>	58
<a href="#">  Constructor patUser</a>	58
<a href="#">  Method addDbc</a>	58
<a href="#">  Method addFieldLocation</a>	59
<a href="#">  Method addGroup</a>	59
<a href="#">  Method addPermission</a>	59
<a href="#">  Method addStats</a>	60
<a href="#">  Method addTable</a>	60
<a href="#">  Method addUser</a>	61
<a href="#">  Method addUserToGroup</a>	61
<a href="#">  Method authenticate</a>	61
<a href="#">  Method clearErrors</a>	62
<a href="#">  Method clearSessionValue</a>	62
<a href="#">  Method convertOptionsToSql</a>	62
<a href="#">  Method countTableEntries</a>	63

<a href="#">Method deleteAllPermissions</a>	63
<a href="#">Method deleteGroup</a>	63
<a href="#">Method deletePermission</a>	64
<a href="#">Method deleteUser</a>	64
<a href="#">Method encryptPasswd</a>	64
<a href="#">Method getAllErrorCodes</a>	65
<a href="#">Method getAllErrorMessages</a>	65
<a href="#">Method getAllErrors</a>	65
<a href="#">Method getAuthFields</a>	65
<a href="#">Method getDbc</a>	66
<a href="#">Method getField</a>	66
<a href="#">Method getGroupData</a>	66
<a href="#">Method getGroupFields</a>	66
<a href="#">Method getGroups</a>	67
<a href="#">Method getHistory</a>	67
<a href="#">Method getJoinedGroups</a>	67
<a href="#">Method getLastError</a>	68
<a href="#">Method getLastErrorCode</a>	68
<a href="#">Method getLastErrorMessage</a>	68
<a href="#">Method getPermissions</a>	68
<a href="#">Method getPermTable</a>	69
<a href="#">Method getPrimaryValue</a>	69
<a href="#">Method getSelfUrl</a>	69
<a href="#">Method getSessionValue</a>	69
<a href="#">Method getTableDef</a>	70
<a href="#">Method getTableInfo</a>	70
<a href="#">Method getUid</a>	70
<a href="#">Method getUserData</a>	71
<a href="#">Method getUsers</a>	71
<a href="#">Method getUsersInGroup</a>	71
<a href="#">Method goHistory</a>	72
<a href="#">Method hasPermission</a>	72
<a href="#">Method identifyGroup</a>	73
<a href="#">Method identifyUser</a>	73
<a href="#">Method isAuthenticated</a>	73
<a href="#">Method isMemberOfGroup</a>	73
<a href="#">Method keepHistory</a>	74
<a href="#">Method logOut</a>	74
<a href="#">Method modifyGroup</a>	75
<a href="#">Method modifyUser</a>	75
<a href="#">Method removeLastError</a>	75
<a href="#">Method removeUserFromGroup</a>	75
<a href="#">Method requireAuthentication</a>	76
<a href="#">Method searchUsers</a>	76
<a href="#">Method sendQuery</a>	77
<a href="#">Method setAuthDbc</a>	77
<a href="#">Method setAuthFields</a>	77
<a href="#">Method setAuthTable</a>	78
<a href="#">Method setCryptFunction</a>	78

<a href="#">Method setError</a>	78
<a href="#">Method setGroupFields</a>	79
<a href="#">Method setGroupRelFields</a>	79
<a href="#">Method setGroupRelTable</a>	79
<a href="#">Method setGroupTable</a>	80
<a href="#">Method setMaxLoginAttempts</a>	80
<a href="#">Method setPermFields</a>	80
<a href="#">Method setPermTable</a>	81
<a href="#">Method setRealm</a>	81
<a href="#">Method setTemplate</a>	81
<a href="#">Method setUnauthorizedURL</a>	82
<a href="#">Method storeSessionValue</a>	82
<a href="#">Method translateErrorCode</a>	82
<a href="#">Method updateStats</a>	83
<b><a href="#">Appendices</a></b>	84
<a href="#">Appendix A - Class Trees</a>	85
<a href="#">patServer</a>	85
<a href="#">patUser</a>	85
<a href="#">patDbc</a>	86



# Package patServer Procedural Elements

## flashchat.php

- **Package** patServer

require\_once ["include/patXMLChatServer.php"](#)*[line 5]*

require\_once ["include/patXMLServer.php"](#)*[line 4]*

require\_once ["include/patServer.php"](#)*[line 3]*

# httpd.php

- **Package** patServer

require\_once ["include/patHTTPServer.php"](#)*[line 4]*

require\_once ["include/patServer.php"](#)*[line 3]*

# jabber.php

- **Package** patServer

require\_once ["include/patJabberServer.php"](#)<sup>[line 3]</sup>

require\_once ["include/patServer.php"](#)<sup>[line 2]</sup>

# patTTT.php

- **Package** patServer

require\_once ["include/patTicTacToe.php"](#)*[line 5]*

require\_once ["include/patXMLServer\\_Dom.php"](#)*[line 4]*

require\_once ["include/patServer.php"](#)*[line 3]*

# telnetchat.php

- **Package** patServer

require\_once ["include/Net\\_Server\\_TelnetChat.php"](#)*[line 3]*

require\_once ["include/patServer.php"](#)*[line 2]*

# telnetchat\_pear.php

- **Package** patServer

require\_once ["include/Net\\_Server\\_TelnetChat.php"](#) *[line 3]*

require\_once **"Net/Server.php"** *[line 2]*

# Package patServer Classes

## Class Net\_Server\_TelnetChat

[line 10]

patTelnetChatServer simple example to demonstrate how patServer is used

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net)>
- **Version** 0.1
- **Access** public

### Net\_Server\_TelnetChat::\$users

*mixed* = array() [line 12]

*integer* function Net\_Server\_TelnetChat::getClientIdByNick(\$nick) [line 163]

#### **Function Parameters:**

- *string* **\$nick** username

### get id of file descriptor by nickname

- **Access** public

# Class patChatServer

*[line 2]*

## patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*  
onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Version** 1.0.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

function patChatServer::onClose(\$clientId) *[line 30]*

### **Function Parameters:**

- **\$clientId**

function patChatServer::onConnect(\$clientId) *[line 35]*

### **Function Parameters:**

- **\$clientId**

function patChatServer::onConnectionRefused(\$clientId) *[line 46]*

### **Function Parameters:**

- **\$clientId**

function patChatServer::onReceiveData(\$clientId, \$data) *[line 10]*

### **Function Parameters:**

- **\$clientId**
- **\$data**

```
function patChatServer::onShutdown() [line 41]
function patChatServer::onStart() [line 5]
```

## Class patFlashChatServer

[line 9]

### patFlashChatServer

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net) >, Gerd Schaufelberger <gerd@php-tools.net>
- **Version** 0.1

### patFlashChatServer::\$dbc

```
user = false [line 22]
```

### patDbc object

- **Var** patDbc

### patFlashChatServer::\$readEndCharacter

```
integer = "\0" [line 34]
```

### end character for socket\_read

### patFlashChatServer::\$user

```
user = false [line 16]
```

### patUser object

- **Var** patUser

**patFlashChatServer::\$users**

*\$users = array() [line 28]*

**data for online users**

- **Var** array()

function patFlashChatServer::doLogin(\$clientId, \$requestParams) *[line 129]*

**Function Parameters:**

- **\$clientId**
- **\$requestParams**

function patFlashChatServer::onClose(\$clientId) *[line 107]*

**Function Parameters:**

- **\$clientId**

function patFlashChatServer::onReceiveRequest(\$clientId, \$requestType, \$requestParams) *[line 44]*

**Function Parameters:**

- *integer* **\$clientId** id of the client that sent the request
- *string* **\$requestType** type of the request
- *array* **\$requestParams** params for the request

**request by client has been received**

- **Access** public

function patFlashChatServer::processTextMessage(\$clientId, \$text) [line 117]

**Function Parameters:**

- **\$clientId**
- **\$text**

function patFlashChatServer::quit(\$clientId, \$requestParams) [line 190]

**Function Parameters:**

- **\$clientId**
- **\$requestParams**

**quit**

## Class patHTTPServer

[line 11]

**patHTTPServer** simple example to demonstrate how patServer is used **Do NOT use**  
in production environments, as it's still quite buggy!

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net)>
- **Version** 0.1
- **Access** public

**patHTTPServer::\$readEndCharacter**

*integer* = "\r\n\r\n" [line 17]

**end character for socket\_read**

*string* function patHTTPServer::getContentType(\$filename, \$filenam) [*line 199*]

**Function Parameters:**

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

**get content type of a file** content types are defined in the httpd.conf file

- **Access public**

function patHTTPServer::onReceiveData(\$clientId, \$data) [*line 43*]

**Function Parameters:**

- *integer* **\$clientId** socket that sent the request
- *string* **\$data** raw request data

**data was received, i.e. HTTP request sent**

function patHTTPServer::onStart() [*line 24*]

**server is started**

read configuration

*array* function patHTTPServer::parsePath(\$path) [*line 165*]

**Function Parameters:**

- *string* **\$path** uri to parse

**parse a request uri**

- **Access public**

array function patHTTPServer::parseRequest(\$request) [line 122]

**Function Parameters:**

- *string* **\$request** raw request data

## parse a http request

- **Access** public

string function patHTTPServer::stripTrailingSlash(\$string, \$path) [line 184]

**Function Parameters:**

- *string* **\$path** path
- **\$string**

## strip trailing slash from a path

- **Access** public

# Class patJabberServer

[line 2]

## patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*

onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Version** 1.0.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

#### **patJabberServer::\$attStack**

*array = array() [line 39]*

#### **stack for attributes**

#### **patJabberServer::\$inMessage**

*array = array() [line 15]*

#### **flag to indicate whether a connection is in a message**

this can be determined, as the client should send wellformed html

#### **patJabberServer::\$inStream**

*array = array() [line 8]*

#### **flag to indicate whether a connection is in stream mode**

#### **patJabberServer::\$readBufferSize**

*integer = 1024 [line 27]*

#### **buffer size for socket\_read**

#### **patJabberServer::\$tagStack**

*array = array() [line 33]*

#### **stack for tags**

#### **patJabberServer::\$xmlParser**

*array = array() [line 21]*

#### **xml parsers for all clients**

function patJabberServer::onClose(\$clientId) [*line 45*]

**Function Parameters:**

- **\$clientId**

**client closes the connection**

function patJabberServer::\_cData(\$parser, \$data) [*line 170*]

**Function Parameters:**

- **\$parser**
- **\$data**

function patJabberServer::\_destroyParser(\$clientId) [*line 180*]

**Function Parameters:**

- **\$clientId**

function patJabberServer::\_endElement(\$parser, \$element) [*line 165*]

**Function Parameters:**

- **\$parser**
- **\$element**

function patJabberServer::\_startElement(\$parser, \$element, \$attributes) [*line 160*]

**Function Parameters:**

- **\$parser**
- **\$element**
- **\$attributes**

**Class patPaintServer**  
[*line 3*]

## patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*

onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Version** 1.0.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

function patPaintServer::onReceiveData(\$clientId, \$data) [*line 5*]

### **Function Parameters:**

- **\$clientId**
- **\$data**

## Class patServer

[*line 17*]

### patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*

onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>
- **Version** 1.0.1

### patServer::\$clientFD

*array = array()* [*line 83*]

**all file descriptors are stored here**

#### **patServer::\$clientInfo**

*array = array() [line 89]*

**needed to store client information**

#### **patServer::\$clients**

*integer = 0 [line 95]*

**amount of clients**

#### **patServer::\$debug**

*boolean = true [line 59]*

**debug mode**

#### **patServer::\$debugDest**

*string = "stdout" [line 71]*

**debug destination (filename or stdout)**

#### **patServer::\$debugMode**

*string = "text" [line 65]*

**debug mode**

#### **patServer::\$domain**

*string = "localhost" [line 29]*

**domain to bind to**

#### **patServer::\$maxClients**

*integer = -1 [line 35]*

**maximum amount of clients**

#### **patServer::\$maxQueue**

*integer = 500 [line 53]*

## maximum of backlog in queue

**patServer::\$null**

*array = array() [line 77]*

## empty array, used for socket\_select

**patServer::\$port**

*integer = 10000 [line 23]*

## port to listen

**patServer::\$readBufferSize**

*integer = 128 [line 41]*

## buffer size for socket\_read

**patServer::\$readEndCharacter**

*integer = "\n" [line 47]*

## end character for socket\_read

Constructor *function patServer::patServer([\$domain = "localhost"], [\$port = 10000]) [line 104]*

### **Function Parameters:**

- *string* **\$domain** domain to bind to
- *integer* **\$port** port to listen to

## create a new socket server

- **Access** public

*int* function **patServer::acceptConnection(&\$socket) [line 286]**

### **Function Parameters:**

- *resource* **&\$socket** socket that received the new connection

## accept a new connection

- **Access** public

function patServer::broadcastData(\$data, [\$exclude = array()]) [*line 411*]

### **Function Parameters:**

- *string* **\$data** data to send
- *array* **\$exclude** client ids to exclude

## send data to all clients

- **Access** public

function patServer::closeConnection(\$id, \$clientID) [*line 333*]

### **Function Parameters:**

- *int* **\$clientID** internal ID of the client
- **\$id**

## close connection to a client

- **Access** public

*array* function patServer::getClientInfo(\$clientId) [*line 433*]

**Function Parameters:**

- *int* **\$clientId** ID of the client

**get current information about a client**

- **Access** public

*int* function patServer::getClients() [*line 379*]

**get current amount of clients**

- **Access** public

*string* function patServer::getLastSocketError(&\$fd) [*line 480*]

**Function Parameters:**

- **&\$fd**

**return string for last socket error**

- **Access** public

*boolean* function patServer::isConnected(\$id) [*line 320*]

**Function Parameters:**

- *integer* **\$id** client id

## check, whether a client is still connected

- **Access** public

function patServer::sendData(\$clientId, \$data, [\$debugData = true]) [*line 392*]

### **Function Parameters:**

- *int* **\$clientId** ID of the client
- *string* **\$data** data to send
- *boolean* **\$debugData** flag to indicate whether data that is written to socket should also be sent as debug message

## send data to a client

- **Access** public

function patServer::setDebugMode(\$debug, [\$dest = "stdout"]) [*line 130*]

### **Function Parameters:**

- *mixed* **\$debug** [text|html|false]
- *string* **\$dest** destination of debug message (stdout to output or filename if log should be written)

## set debug mode

- **Access** public

function patServer::setMaxClients(\$maxClients) [*line 118*]

**Function Parameters:**

- *int* **\$maxClients**

**set maximum amount of simultaneous connections**

- **Access public**

function patServer::shutDown() [*line 354*]

**shutdown server**

- **Access public**

function patServer::start(\$maxClients) [*line 149*]

**Function Parameters:**

- *int* **\$maxClients**

**start the server**

- **Access public**

# Class patTelnetChatServer

*[line 10]*

**patTelnetChatServer** simple example to demonstrate how patServer is used

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net)>
- **Version** 0.1
- **Access** public

## **patTelnetChatServer::\$users**

*mixed = array() [line 12]*

*integer function patTelnetChatServer::getClientIdByNick(\$nick) [line 163]*

### **Function Parameters:**

- *string* **\$nick** username

## **get id of file descriptor by nickname**

- **Access** public

# Class patTicTacToe

*[line 2]*

## **patServer**

PHP socket xml server base class    Events that can be handled:    \* onStart    \*  
onConnect    \* onConnectionRefused    \* onClose    \* onShutdown    \* onReceiveRequest

Methods used to send responses:    \* sendResponse    \* broadcastResponse

- **Package** patServer
- **Version** 0.1
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net)>, Gerd Schaufelberger <gerd@php-tools.net>

#### patTicTacToe::\$candidates

```
array = array(
    array( 0, 1, 2
),array(3,4,5),array(6,7,8),array(0,3,6),array(1,4,7),array(2,5,8),array(0,4,8),array(2,4,6)) [line 49]
```

#### win candidates

#### patTicTacToe::\$emptyField

```
array = array(
    array( "", "", "" ),array("", "", ""),array("", "", "")) [line 38]
```

#### template for a new game

#### patTicTacToe::\$gameId

```
mixed = 1 [line 20]
```

#### id of the next game

```
var integer $gameId
```

#### patTicTacToe::\$games

```
mixed = array() [line 14]
```

#### games that are in use

```
var array $games
```

#### patTicTacToe::\$players

```
array = array() [line 26]
```

#### all players

#### patTicTacToe::\$readEndCharacter

*integer = "\0" [line 8]*

## **end character for socket\_read**

### **patTicTacToe::\$waitingPlayers**

*array = array() [line 32]*

## **users that are waiting to play (normally only one)**

*function patTicTacToe::checkForNewGames() [line 124]*

*function patTicTacToe::createNewGame(\$player1, \$player2) [line 143]*

### **Function Parameters:**

- *int \$player1*
- *int \$player2*

## **create a new game**

*function patTicTacToe::endGame(\$gameId) [line 423]*

### **Function Parameters:**

- *int \$gameId*

## **end a game**

*int function patTicTacToe::getGameId(\$clientId) [line 357]*

### **Function Parameters:**

- *integer \$clientId*

## **get id of a game for a player**

*integer function patTicTacToe::getOpponentId(\$clientId) [line 370]*

### **Function Parameters:**

- *integer \$clientId*

## get id of the opponent a player

function patTicTacToe::onClose(\$clientId) [*line 385*]

### **Function Parameters:**

- *integer* **\$clientId**

## client closed connection

function patTicTacToe::onReceiveRequest(\$clientId, \$requestType, \$requestParams) [*line 69*]

### **Function Parameters:**

- *integer* **\$clientId**
- *string* **\$requestType**
- *array* **\$requestParams**

## request by flash received

function patTicTacToe::putOnHold(\$clientId) [*line 441*]

### **Function Parameters:**

- *\$clientId* **\$clientId**

## put player on hold

function patTicTacToe::sendResponseToPlayers(\$gameId, \$type, \$params, \$responseType, \$responseParams) [*line 224*]

### **Function Parameters:**

- *integer* **\$gameId**
- *string* **\$responseType**
- *array* **\$responseParams**
- **\$type**
- **\$params**

## send a response to both players of a game

function patTicTacToe::startGame(\$gameId) [*line 174*]

**Function Parameters:**

- *integer* **\$gameId**

**start a game**

## Class patXMLChatServer

[*line 3*]

### patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*  
onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Version** 1.0.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

### patXMLChatServer::\$users

*mixed = array()* [*line 5*]

function patXMLChatServer::doLogin(\$clientId, \$userData) [*line 52*]

**Function Parameters:**

- **\$clientId**
- **\$userData**

function patXMLChatServer::onClose(\$clientId) [*line 42*]

**Function Parameters:**

- **\$clientId**

function patXMLChatServer::onReceiveData(\$clientId, \$data) [*line 7*]

**Function Parameters:**

- **\$clientId**
- **\$data**

## Class patXMLServer

[*line 2*]

### patServer

PHP socket server base class      Events that can be handled:      \* onStart      \*  
onConnect      \* onConnectionRefused      \* onClose      \* onShutdown      \* onReceiveData

- **Package** patServer
- **Version** 1.0.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

### patXMLServer::\$parser

*mixed* = [*line 4*]

function patXMLServer::buildXML(\$root, \$values) [*line 55*]

**Function Parameters:**

- **\$root**
- **\$values**

function patXMLServer::characterData(\$parser, \$data) [*line 19*]

**Function Parameters:**

- **\$parser**
- **\$data**

function patXMLServer::endElement(\$parser, \$name) *[line 24]*

**Function Parameters:**

- **\$parser**
- **\$name**

function patXMLServer::onStart() *[line 6]*

function patXMLServer::parseXML(\$xml) *[line 33]*

**Function Parameters:**

- **\$xml**

function patXMLServer::startElement(\$parser, \$name, \$attributes) *[line 13]*

**Function Parameters:**

- **\$parser**
- **\$name**
- **\$attributes**

## Class patXMLServer\_Dom

*[line 21]*

### patServer

PHP socket xml server base class    Events that can be handled:    \* onStart    \*  
onConnect    \* onConnectionRefused    \* onClose    \* onShutdown    \* onReceiveRequest

Methods used to send responses:    \* sendResponse    \* broadcastResponse

- **Package** patServer
- **Author** Stephan Schmidt < [schst@php-tools.net](mailto:schst@php-tools.net)>, Gerd Schaufelberger <gerd@php-tools.net>
- **Version** 0.1

```
function patXMLServer_Dom::broadcastResponse($responseType, $responseParams, [$exclude = array()], $data) [line 83]
```

**Function Parameters:**

- *string* **\$data** data to send
- *array* **\$exclude** client ids to exclude
- **\$responseType**
- **\$responseParams**

## send response to all clients

- **Access** public

```
string function patXMLServer_Dom::encodeResponse($responseType, $responseParams) [line 97]
```

**Function Parameters:**

- *string* **\$responseType** type of response
- *array* **\$responseParams** all params

## encode a request

- **Access** public

```
boolean function patXMLServer_Dom::sendResponse($clientId, $responseType, $responseParams) [line 70]
```

**Function Parameters:**

- *integer* **\$clientId** id of the client to that the response should be sent
- *string* **\$responseType** type of response
- *array* **\$responseParams** all params

## send a response

- **Access public**



# Package patDbc Procedural Elements

## patDbc.php

- **Package** patDbc
- **Version** 0.81
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>, Gerd Schauffelberger <gerd@php-tools.de>

```
patDBC_CLOSED = 0 [line 8]
patDBC_OPEN = 1 [line 9]
patDBC_TYPEASSOC = 3 [line 11]
patDBC_TYPEBOTH = 2 [line 10]
patDBC_TYPENUM = 4 [line 12]
```

# Package patDbc Classes

## Class patDbcResult

*[line 20]*

**generic result object, will be extended by result objects for databases**

- **Package** patDbc
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>, Gerd Schaufelberger <gerd@php-tools.de>

### patDbcResult::\$id

*int = [line 25]*

- **Var** ressource id of the result object

function patDbcResult::dump() *[line 118]*

**dump the result**

- **Access** public

function patDbcResult::dumpRow(\$row) [*line 134*]

**Function Parameters:**

- *array* **\$row** array containing one row

**dump one row**

*array* function patDbcResult::fetch\_array([\$result\_type = patDBC\_TYPEBOTH]) [*line 55*]

**Function Parameters:**

- *int* **\$result\_type** type of result (patDBC\_TYPEBOTH, patDBC\_TYPEASSOC, patDBC\_TYPENUM)

**fetch one row of the result as associative array**

- **Access** public

*array* function patDbcResult::fetch\_row() [*line 43*]

**fetch one row of the result**

- **Access** public

*boolean* function patDbcResult::free() [*line 108*]

**free used memory used by the result**

- **Access** public

array function patDbcResult::get\_result([\$result\_type = patDBC\_TYPEBOTH]) [*line 67*]

**Function Parameters:**

- *int* **\$result\_type** type of result (patDBC\_TYPEBOTH, patDBC\_TYPEASSOC, patDBC\_TYPENUM)

**get the complete result as two dimensional array**

- **Access public**

*int* function patDbcResult::num\_fields() [*line 97*]

**get the number of fields in the result**

- **Access public**

*int* function patDbcResult::num\_rows() [*line 86*]

**get the number of rows in the result**

- **Access public**

function patDbcResult::setIdentifier(\$id) [*line 32*]

**Function Parameters:**

- *int* **\$id** result identifier

**set the result identifier**

# Class patMySqlDbc

[line 296]

**MySql Abstraction** provides commonly needed mysql functions

- **Package** patDbc
- **Access** public

**patMySqlDbc::\$dbhost**

*mixed = [line 298]*

**patMySqlDbc::\$dbid**

*mixed = [line 299]*

**patMySqlDbc::\$dbname**

*mixed = [line 298]*

**patMySqlDbc::\$dbpass**

*mixed = [line 298]*

**patMySqlDbc::\$dbuser**

*mixed = [line 298]*

**patMySqlDbc::\$status**

*mixed = [line 300]*

Constructor function patMySqlDbc::patMySqlDbc(\$dbhost, \$dbname, \$dbuser, \$dbpass) [line 311]

**Function Parameters:**

- *string* **\$dbhost** hostname
- *string* **\$dbname** name of the database
- *string* **\$dbuser** name of the user
- *string* **\$dbpass** password for the user

**Create new MySql database connectivity**

- **Access** public

*int* function patMySqlDbc::affected\_rows() [*line 407*]  
**count the number of affected rows**

- **Access** public

function patMySqlDbc::close() [*line 360*]  
**Close the MySql connection**

- **Access** public

*boolean* function patMySqlDbc::connect([\$die = true], [\$persistent = false]) [*line 329*]  
**Function Parameters:**

- *boolean* **\$die** if set to true, the script will exit if the connection could not be established
- *boolean* **\$persistent** if set to true, a persistent connection will be opened

## **Open the MySql connection**

- **Access** public

*bool* function patMySqlDbc::create\_db([\$database\_name = ""]) [*line 434*]  
**Function Parameters:**

- *string* **\$database\_name** name of the db to create, if no name is given the name of the db used for the connection will be used

## create a database

- **Access public**

*bool* function patMySqlDbc::drop\_db([\$database\_name = ""]) [*line 419*]

### **Function Parameters:**

- *string* **\$database\_name** name of the db to drop, if no name is given the db of the connection will be dropped

## drop a database

- **Access public**

*array* function patMySqlDbc::get\_fields(\$table) [*line 373*]

### **Function Parameters:**

- *mixed* **\$table** name of the table or array containing names of several tables

## get fieldnames of a mysql table

- **Access public**

*int* function patMySqlDbc::insert\_id() [*line 396*]  
**get the last insert id for this connection**

- **Access** public

*object patDbcResult* function patMySqlDbc::query(\$query) [*line 449*]  
**Function Parameters:**

- *string* **\$query** query that should be send

## Send a query to mysql

- **Access** public

# Class patMySqlResult

[*line 153*]

**mysql result object** result object returned by the query() method of the patMySqlDbc

- **Package** patDbc
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>

function patMySqlResult::data\_seek(\$row\_number) [*line 226*]  
**Function Parameters:**

- *int* **\$row\_number** row number

### move the internal row pointer of the result

- **Access** public

*array* function `patMySqlResult::fetch_array([$result_type = patDBC_TYPEBOTH])` [*line 173*]

#### **Function Parameters:**

- *int* **\$result\_type** type of result (patDBC\_TYPEBOTH, patDBC\_TYPEASSOC, patDBC\_TYPENUM)

### fetch one row of the result as associative array

- **Access** public

*object field* function `patMySqlResult::fetch_field([$offset = 0])` [*line 238*]

#### **Function Parameters:**

- *integer* **\$offset** field offset (starting with 0)

### fetch information about a field of the result

- **Access** public

*object result* function `patMySqlResult::fetch_object([$result_type = patDBC_TYPEBOTH])` [*line 189*]

**Function Parameters:**

- *integer* **\$result\_type** type of result (`patDBC_TYPEBOTH`, `patDBC_TYPEASSOC`, `patDBC_TYPENUM`)

**fetch one row of the result as an object**

- **Access public**

*array* function `patMySqlResult::fetch_row()` [*line 161*]

**fetch one row of the result**

- **Access public**

*int* function `patMySqlResult::field_len($offset)` [*line 262*]

**Function Parameters:**

- *int* **\$offset** field offset (starting with 0)

**returns the length of the specified field**

- **Access public**

*string* function `patMySqlResult::field_name($offset)` [*line 250*]

**Function Parameters:**

- *integer* **\$offset** field offset (starting with 0)

## get the name of a field

- **Access** public

function patMySqlResult::field\_seek(\$offset) [*line 273*]

### **Function Parameters:**

- *int* **\$offset** field offset (starting with 0)

**Seeks to the specified field offset.**

- **Access** public

*boolean* function patMySqlResult::free() [*line 283*]

**free used memory used by the result**

- **Access** public

*int* function patMySqlResult::num\_fields() [*line 215*]

**get the number of fields in the result**

- **Access** public

*int* function patMySqlResult::num\_rows() [*line 204*]

**get the number of rows in the result**

- **Access public**



# Package patUser Procedural Elements

## patUser.php

- **Package** patUser

patUSER\_ALREADY\_JOINED\_GROUP = 1050 *[line 155]*  
**error code: user already is in group**

- **Access** public

patUSER\_COULD\_NOT\_IDENTIFY\_LINE = 110 *[line 85]*  
**error code: could not identify line in table**

- **Access** public

patUSER\_DELETE\_NOT\_ALLOWED = 121 *[line 99]*  
**error code: delete is not allowed**

- **Access** public

patUSER\_GROUP\_ALREADY\_EXISTS = 1012 *[line 141]*  
**error code: group already exists (when adding a group)**

- **Access public**

patUSER\_INSERT\_NOT\_ALLOWED = 120 *[line 92]*  
**error code: insert is not allowed**

- **Access public**

patUSER\_LOGIN\_DISABLED = 14 *[line 50]*  
**error code: login for this user was disabled**

- **Access public**

patUSER\_NEED\_GID = 1020 *[line 148]*  
**error code: function requires group id**

- **Access public**

patUSER\_NEED\_GROUPNAME = 1010 *[line 134]*  
**error code: function requires a group name**

- **Access public**

patUSER\_NEED\_ID = 1060 *[line 169]*

**error code: function requires user or group id**

- **Access public**

patUSER\_NEED\_ID\_TYPE = 1061 *[line 176]*

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

- **Access public**

patUSER\_NEED\_PASSWD = 11 *[line 29]*

**error code: function requires a password**

- **Access public**

patUSER\_NEED\_UID = 20 *[line 57]*

**error code: function requires a user id**

- **Access public**

patUSER\_NEED\_USERNAME = 10 *[line 22]*

**error code: function requires a user name**

- **Access** public

patUSER\_NOT\_IN\_GROUP = 1051 *[line 162]*

**error code: user is not in group**

- **Access** public

patUSER\_NO\_DATA\_CHANGED = 130 *[line 106]*

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

- **Access** public

patUSER\_NO\_DATA\_GIVEN = 30 *[line 64]*

**error code: function requires data**

- **Access** public

patUSER\_NO\_DB\_RESULT = 2000 *[line 183]*

**error code: query had no result**

- **Access** public

patUSER\_NO\_GROUP\_FOUND = 1001 *[line 120]*  
**error code: no group matched the query**

- **Access public**

patUSER\_NO\_PRIMARY\_FOUND = 100 *[line 71]*  
**error code: no primary key was found**

- **Access public**

patUSER\_NO\_UNIQUE\_GROUP\_FOUND = 1002 *[line 127]*  
**error code: no unique group matched the query**

- **Access public**

patUSER\_NO\_UNIQUE\_PRIMARY\_FOUND = 101 *[line 78]*  
**error code: data matched more than one row**

- **Access public**

patUSER\_NO\_UNIQUE\_USER\_FOUND = 2 *[line 15]*  
**error code: more than one user matror the que y**

- **Access public**

patUSER\_NO\_USER\_FOUND = 1 *[line 8]*  
**error code: no user matched the query**

- **Access public**

patUSER\_PASSWD\_MISMATCH = 13 *[line 43]*  
**error code: password incorrect**

- **Access public**

patUSER\_TABLE\_DOES\_NOT\_EXIST = 140 *[line 113]*  
**error code: table does not exist**

- **Access public**

patUSER\_USER\_ALREADY\_EXISTS = 12 *[line 36]*  
**error code: user already exists**

- **Access public**

# Package patUser Classes

## Class patUser

*[line 193]*

patUser \$Id: patUser.php 2 2004-03-02 23:38:08Z schst \$

- **Package** patUser
- **Version** 2.1.1
- **Author** Stephan Schmidt < [schst@php-tools.de](mailto:schst@php-tools.de)>, Gerd Schaufelberger <gerd@php-tools.de>

### patUser::\$actionVar

*string = "patUserAction" [line 372]*

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

### patUser::\$authenticated

*bool = false [line 280]*

**state of user/session: authenticated or not**

### patUser::\$authFields

```
array = array( "primary" => "uid",  
              "username" => "username",  
              "passwd" => "passwd" ) [line 286]
```

**Fieldnames in the authentication table**

## patUser::\$authTable

```
string = "users" [line 274]
```

**Table that stores the authentication data**

## patUser::\$cryptFunction

```
string = false [line 426]
```

**name of function, that should be used for passwd encryption**

## patUser::\$dbcs

```
array = array() [line 396]
```

**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 212]
```

**error messages**

#### **patUser::\$errors**

```
array = array() [line 408]
```

**all codes of the errors that happened while processing**

#### **patUser::\$fieldLocs**

```
array = array() [line 268]
```

**locations (table/fieldname) of fields**

#### **patUser::\$groupFields**

```
array = array( "primary" => "gid",  
              "name" => "name" ) [line 300]
```

**Fieldnames in the group table**

#### **patUser::\$groupTable**

```
string = "groups" [line 294]
```

**Table that stores the group data**

#### **patUser::\$ignoreVars**

```
array = array( "patUserAction" ) [line 420]
```

**variable names that should be ignored by getSelfUrl()**

#### **patUser::\$loginTemplate**

```
string = "patUserLogin.tmpl" [line 378]
```

**filename of the template used for login**

#### **patUser::\$maxLoginAttempts**

```
integer = 0 [line 250]
```

**maximum login attempts for a session**

#### **patUser::\$options**

```
array = array() [line 262]
```

## options of the user class

### patUser::\$outputErrors

```
array = array() [line 414]
```

## all codes of the errors that will be outputted

### patUser::\$permFields

```
array = array( "id" => "id",  
              "id_type" => "id_type",  
              "perms" => "perms" ) [line 326]
```

## Fieldnames in the permissions table

### patUser::\$perms

```
array = array( 1 => "read",  
              2 => "delete",  
              4 => "modify",  
              8 => "add" ) [line 334]
```

## Possible permissions

### patUser::\$permsConv

```
array = array() [line 342]
```

## table to convert permissions

### patUser::\$permTable

```
string = "permissions" [line 320]
```

## Table that stores the permissions

### patUser::\$realm

```
string = "patUser protected area" [line 244]
```

## default realm for HTTP authentication

### patUser::\$relFields

```
array = array( "uid" => "uid",  
              "gid" => "gid" ) [line 313]
```

## Fieldnames of the user-group relation table

### patUser::\$relTable

```
string = "usergroups" [line 307]
```

### Table that stores the user - group relations

### patUser::\$sessionVar

```
string = "patUserData" [line 360]
```

### name of the global variable used for sessions

### patUser::\$stats

```
array = array() [line 348]
```

### all statistic options

### patUser::\$systemVars

```
array = array(  
    "appName" => "patUser",  
    "appVersion" => "2.1.1",  
    "author" => array(  
        "Stephan Schmidt <schst@php-tools.de>",  
        "Gerd Schaufelberger <gerd@php-tools.de>"  
    )) [line 199]
```

### information about the project

### patUser::\$tables

```
array = array() [line 402]
```

### all tables that are used

### patUser::\$tmpl

```
boolean = false [line 366]
```

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

### patUser::\$unauthorizedTemplate

```
string = "patUserUnauthorized.tmpl" [line 384]
```

## filename of the template used for unauthorized users

**patUser::\$unauthorizedURL**

*string = false [line 390]*

## URL to redirect unauthorized users to

**patUser::\$useSessions**

*boolean = false [line 354]*

## flag to indicate whether sessions are used

**patUser::\$useTemplate**

*string = false [line 256]*

## flag to indicate, whether template is used for output

Constructor function patUser::patUser([\$useSessions = true], [\$sessionVar = "patUserData"]) [line 435]

### **Function Parameters:**

- *boolean* **\$useSessions** flag to indicate that sessions should be used
- *string* **\$sessionVar** name of the session var used for sessions

## create new user object

- **Access public**

function patUser::addDbc(\$name, &\$dbc) [line 1335]

### **Function Parameters:**

- *string* **\$name** unique name to access the dbc
- *object* **patDbc &\$dbc** patDbc object

## add a dbc you can use as many dbcs as you like to access data stored in different

## databases

- **Access** public

function patUser::addFieldLocation(\$name, [\$table = "authTable"], [\$field = ""]) [*line 694*]

**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 2163*]

**Function Parameters:**

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

### adds a group

- **Access** public

*bool* function patUser::addPermission(\$perm) [*line 2602*]

**Function Parameters:**

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

## add permission(s)

- **Access** public

function patUser::addStats(\$statistic, [\$field = ""]) [*line 678*]

### **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**

- **Access** public

function patUser::addTable(\$name, \$tabledef) [*line 1354*]

### **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 1110*]

**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**

- **Access** public

*bool* function patUser::addUserToGroup(\$relation) [*line 2309*]

**Function Parameters:**

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

### **add a user to a group**

- **Access** public

*mixed* function patUser::authenticate([\$data = array()]) [*line 724*]

**Function Parameters:**

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

## authenticate a user

- **Access** public

function patUser::clearErrors() [*line 3196*]

### clear all errors

- **Access** public

*bool* function patUser::clearSessionValue(\$name) [*line 1205*]

#### **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**

- **Access** public

*string* function patUser::convertOptionsToSql([\$options = array()]) [*line 3104*]

#### **Function Parameters:**

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

## convert options to sql

access private

function patUser::countTableEntries(\$table, [\$suid = 0]) [*line 1817*]

**Function Parameters:**

- *string* **\$table** name of the table
- *integer* **\$suid** 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 2679*]

**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 2209*]

**Function Parameters:**

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

**delete an existing group**

- **Access** public

*bool* function patUser::deletePermission(\$perm) [*line 2620*]

**Function Parameters:**

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

### **delete permission(s)**

- **Access** public

function patUser::deleteUser([\$options = array()]) [*line 1655*]

**Function Parameters:**

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

### **delete (data of) an existing user**

- **Access** public

*string* function patUser::encryptPasswd(\$passwd) [*line 3318*]

**Function Parameters:**

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

### **encrypt a password using the specified encryption function**

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

*array function patUser::getAllErrorCodes() [line 3241]*

### **get all error codes**

- Access public

*array function patUser::getAllErrorMessage() [line 3252]*

### **get all error messages**

- Access public

*array function patUser::getAllErrors() [line 3267]*

### **get all error codes AND messages**

- Access public

*array function patUser::getAuthFields() [line 547]*

### **get fieldnames that contain auth info**

- Access public

object *patDbc* function *patUser::getDbc(\$table)* [*line 3159*]

**Function Parameters:**

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

### get the dbc object for a table

mixed function *patUser::getField(\$field, [\$uid = 0])* [*line 1980*]

**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 2049*]

**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 585*]

### get fieldnames that contain group info

- **Access** public

function patUser::getGroups([\$fields = array()], [\$clause = array()], [\$options = array()]) [*line 2030*]

**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 1288*]

**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 2236*]

**Function Parameters:**

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

## get joined groups

- **Access** public

*array* function patUser::getLastError() [*line 3229*]  
**get the last error code AND message**

- **Access** public

*integer* function patUser::getLastErrorCode() [*line 3207*]  
**get the last error code**

- **Access** public

*string* function patUser::getLastErrorMessage() [*line 3218*]  
**get the last error message**

- **Access** public

*array* function patUser::getPermissions([\$clause = array()], [\$type = "both"]) [*line 2450*]  
**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 )

## get all permissions of a user

- **Access** public

*string* function patUser::getPermTable() [*line 633*]

## get tablename that contains permissions

- **Access** public

function patUser::getPrimaryValue(\$uid, \$table, \$data) [*line 1773*]

### **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

function patUser::getSelfUrl() [*line 3127*]

*mixed* function patUser::getSessionValue(\$name) [*line 1224*]

### **Function Parameters:**

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

**get a value that was stored in the session with \$useSessions = true**      **may only be used if patUser was called**

- **Access public**

*array* function patUser::getTableDef(\$name) [*line 1369*]

**Function Parameters:**

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

**get table definitions**

- **Access public**

*array* function patUser::getTableInfo(\$tablename) [*line 2944*]

**Function Parameters:**

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

**get information about table fields**    **like DESCRIBE [table]**

- **Access public**

*int* function patUser::getUid() [*line 798*]

**return** **the user id of the current user**

- **Access public**

*array* function patUser::getUserData([\$options = array()], [\$clause = array()]) [*line 1384*]

**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 1435*]

**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 2270*]

**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 1314]

### **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 2515]

### **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, wether user has a permission

- **Access** public

function patUser::identifyGroup(\$fields, [\$createError = true]) [*line 2076*]

**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 1458*]

**Function Parameters:**

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

## identify a user by certain fields

- **Access public**

*bool* function patUser::isAuthenticated() [*line 772*]

**check, wether 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 2429*]

**Function Parameters:**

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

## check, if a user is in a group

- **Access** public

*bool* function patUser::keepHistory(\$amount, [\$title = ""]) [*line 1244*]

### **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 1077*]

### **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 2115*]

**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 1520*]

**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 3186*]

## remove the last error from the list

- **Access** public

*bool* function patUser::removeUserFromGroup(\$relation) [*line 2358*]

**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 855*]

### **Function Parameters:**

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

## require an authenticated user

if no authenticated user is found, a login screen will be displayed (needs a patTemplate object) or exit will be forced if displayLogin is set to false

- **Access** public

*array* function patUser::searchUsers(\$criterias, [\$options = array()]) [*line 1851*]

### **Function Parameters:**

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

## Search for users matching certain criterias

- **Access** public

object *patDbcResult* function patUser::sendQuery(\$dbc, \$query) [line 3083]

**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

function patUser::setAuthDbc(&\$dbc) [line 510]

**Function Parameters:**

- *object* **patDbc** &**\$dbc** patDbc Object

**set dbc that contains auth info**

- **Access** public

function patUser::setAuthFields(\$fields) [line 532]

**Function Parameters:**

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

## set fieldnames that contain auth info

- **Access** public

function patUser::setAuthTable(\$table) [*line 521*]

### **Function Parameters:**

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

## set tablename that contains auth info

- **Access** public

function patUser::setCryptFunction([\$cryptFunction = "crypt"]) [*line 708*]

### **Function Parameters:**

- *string* **\$cryptFunction** name of the encryption function

## set function that is used for passwd encryption

- **Access** public

function patUser::setError(\$error) [*line 3176*]

### **Function Parameters:**

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

## set error code

function patUser::setGroupFields(\$fields) [*line 570*]

### **Function Parameters:**

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

## set fieldnames that contain group info

- **Access** public

function patUser::setGroupRelFields(\$fields) [*line 607*]

### **Function Parameters:**

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

## set fieldnames that are used in the user group relations

- **Access** public

function patUser::setGroupRelTable(\$table) [*line 596*]

### **Function Parameters:**

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

## set tablename that contains user group relations

- **Access** public

function patUser::setGroupTable(\$table) [*line 559*]

**Function Parameters:**

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

### **set tablename that contains group info**

- **Access** public

function patUser::setMaxLoginAttempts(\$maxLoginAttempts) [*line 488*]

**Function Parameters:**

- *integer* **\$maxLoginAttempts**

### **set maximum amount of login attempts**

- **Access** public

function patUser::setPermFields(\$fields) [*line 645*]

**Function Parameters:**

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

### **set fieldnames that are used in the permissions**

- **Access** public

function patUser::setPermTable(\$table) [*line 622*]

**Function Parameters:**

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

**set tablename that contains permissions**

- **Access** public

function patUser::setRealm(\$realm) [*line 477*]

**Function Parameters:**

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

**set authentication realm**

- **Access** public

function patUser::setTemplate(&\$tmpl) [*line 661*]

**Function Parameters:**

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

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

- **Access public**

function patUser::setUnauthorizedURL(\$url) [*line 499*]

**Function Parameters:**

- *string* **\$url**

**set URL to redirect unauthorized users to**

- **Access public**

bool function patUser::storeSessionValue(\$name, \$val) [*line 1188*]

**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**

- **Access public**

string function patUser::translateErrorCode(\$code) [*line 3285*]

**Function Parameters:**

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

## translate an error code

- **Access** public

function patUser::updateStats() [*line 2856*]

### **update all statistics**

- **Access** public

# Appendices

# Appendix A - Class Trees

## Package patServer

### Net\_Server\_TelnetChat

- Net\_server
  - [Net\\_Server\\_TelnetChat](#)

### patServer

- [patServer](#)
  - [patChatServer](#)
  - [patHTTPServer](#)
  - [patJabberServer](#)
  - [patTelnetChatServer](#)
  - [patXMLServer](#)
    - [patPaintServer](#)
    - [patXMLChatServer](#)
  - [patXMLServer\\_Dom](#)
    - [patFlashChatServer](#)
    - [patTicTacToe](#)

## Package patUser

### patUser

- [patUser](#)

## Package patDbc

### patDbcResult

- [patDbcResult](#)
  - [patMySqlResult](#)

### patMySqlDbc

- [patMySqlDbc](#)

# Index

## C

<a href="#">constructor patUser::patUser()</a> . . . . .	58
<i>create new user object</i>	
<a href="#">constructor patMySqlDbc::patMySqlDbc()</a> . . . . .	38
<i>Create new MySql database connectivity</i>	
<a href="#">constructor patServer::patServer()</a> . . . . .	19
<i>create a new socket server</i>	

## F

<a href="#">flashchat.php</a> . . . . .	2
---	---

## H

<a href="#">httpd.php</a> . . . . .	3
-------------------------------------	---

## J

<a href="#">jabber.php</a> . . . . .	4
--------------------------------------	---

## N

<a href="#">Net_Server_TelnetChat::getClientIdByNick()</a> . . . . .	8
<i>get id of file descriptor by nickname</i>	
<a href="#">Net_Server_TelnetChat::\$users</a> . . . . .	8
<a href="#">Net_Server_TelnetChat</a> . . . . .	8
<i>patTelnetChatServer</i>	
<i>simple example to demonstrate how patServer is used</i>	

## P

<a href="#">patUser::\$options</a> . . . . .	55
<i>options of the user class</i>	
<a href="#">patUser::\$maxLoginAttempts</a> . . . . .	55
<i>maximum login attempts for a session</i>	
<a href="#">patUser::\$loginTemplate</a> . . . . .	55
<i>filename of the template used for login</i>	
<a href="#">patUser::\$outputErrors</a> . . . . .	56
<i>all codes of the errors that will be outputted</i>	
<a href="#">patUser::\$permFields</a> . . . . .	56

<i>Fieldnames in the permissions table</i>	56
<a href="#">patUser::\$permsConv</a>	56
<i>table to convert permissions</i>	
<a href="#">patUser::\$perms</a>	56
<i>Possible permissions</i>	
<a href="#">patUser::\$ignoreVars</a>	55
<i>variable names that should be ignored by getSelfUrl()</i>	
<a href="#">patUser::\$groupTable</a>	55
<i>Table that stores the group data</i>	
<a href="#">patUser::\$dbcs</a>	54
<i>all dbcs</i>	
<a href="#">patUser::\$cryptFunction</a>	54
<i>name of function, that should be used for passwd encryption</i>	
<a href="#">patUser::\$authTable</a>	54
<i>Table that stores the authentication data</i>	
<a href="#">patUser::\$errorMessages</a>	54
<i>error messages</i>	
<a href="#">patUser::\$errors</a>	55
<i>all codes of the errors that happened while processing</i>	
<a href="#">patUser::\$groupFields</a>	55
<i>Fieldnames in the group table</i>	
<a href="#">patUser::\$fieldLocs</a>	55
<i>locations (table/fieldname) of fields</i>	
<a href="#">patUser::\$permTable</a>	56
<i>Table that stores the permissions</i>	
<a href="#">patUser::\$realm</a>	56
<i>default realm for HTTP authentication</i>	
<a href="#">patUser::\$addDbc()</a>	58
<i>add a dbc</i>	
<i>you can use as many dbcs as you like to access data stored in different databases</i>	
<a href="#">patUser::\$useTemplate</a>	58
<i>flag to indicate, whether template is used for output</i>	
<a href="#">patUser::\$useSessions</a>	58
<i>flag to indicate whether sessions are used</i>	
<a href="#">patUser::\$addFieldLocation()</a>	59
<i>set the location of a field</i>	
<a href="#">patUser::\$addGroup()</a>	59
<i>adds a group</i>	
<a href="#">patUser::\$addStats()</a>	60
<i>add a statistic option</i>	
<i>stats include: first_login, last_login, count_logins, count_pages, time_online</i>	
<a href="#">patUser::\$addPermission()</a>	59
<i>add permission(s)</i>	
<a href="#">patUser::\$unauthorizedURL</a>	58
<i>URL to redirect unauthorized users to</i>	
<a href="#">patUser::\$unauthorizedTemplate</a>	57
<i>filename of the template used for unauthorized users</i>	
<a href="#">patUser::\$sessionVar</a>	57
<i>name of the global variable used for sessions</i>	
<a href="#">patUser::\$relTable</a>	57
<i>Table that stores the user - group relations</i>	
<a href="#">patUser::\$relFields</a>	56
<i>Fieldnames of the user-group relation table</i>	

<a href="#">patUser::\$stats</a>	57
<i>all statistic options</i>	
<a href="#">patUser::\$systemVars</a>	57
<i>information about the project</i>	
<a href="#">patUser::\$tmpl</a>	57
<i>variable to store the patTemplate object (false if no template is used)</i>	
<a href="#">patUser::\$tables</a>	57
<i>all tables that are used</i>	
<a href="#">patUser::\$authFields</a>	53
<i>Fieldnames in the authentication table</i>	
<a href="#">patUser::\$authenticated</a>	53
<i>state of user/session: authenticated or not</i>	
<a href="#">patUSER LOGIN DISABLED</a>	48
<i>error code: login for this user was disabled</i>	
<a href="#">patUSER INSERT NOT ALLOWED</a>	48
<i>error code: insert is not allowed</i>	
<a href="#">patUSER GROUP ALREADY EXISTS</a>	48
<i>error code: group already exists (when adding a group)</i>	
<a href="#">patUSER NEED GID</a>	48
<i>error code: function requires group id</i>	
<a href="#">patUSER NEED GROUPNAME</a>	48
<i>error code: function requires a group name</i>	
<a href="#">patUSER NEED ID TYPE</a>	49
<i>error code: function requires type of supplied id (user or group)</i>	
<a href="#">patUSER NEED ID</a>	49
<i>error code: function requires user or group id</i>	
<a href="#">patUSER DELETE NOT ALLOWED</a>	47
<i>error code: delete is not allowed</i>	
<a href="#">patUSER COULD NOT IDENTIFY LINE</a>	47
<i>error code: could not identify line in table</i>	
<a href="#">patMySqlResult::free()</a>	44
<i>free used memory used by the result</i>	
<a href="#">patMySqlResult::field_seek()</a>	44
<i>Seeks to the specified field offset.</i>	
<a href="#">patMySqlResult::field_name()</a>	43
<i>get the name of a field</i>	
<a href="#">patMySqlResult::num_fields()</a>	44
<i>get the number of fields in the result</i>	
<a href="#">patMySqlResult::num_rows()</a>	45
<i>get the number of rows in the result</i>	
<a href="#">patUSER ALREADY JOINED GROUP</a>	47
<i>error code: user already is in group</i>	
<a href="#">patUser.php</a>	47
<a href="#">patUSER NEED PASSWD</a>	49
<i>error code: function requires a password</i>	
<a href="#">patUSER NEED UID</a>	49
<i>error code: function requires a user id</i>	
<a href="#">patUSER PASSWD MISMATCH</a>	52
<i>error code: password incorrect</i>	
<a href="#">patUSER NO USER FOUND</a>	52
<i>error code: no user matched the query</i>	
<a href="#">patUSER NO UNIQUE USER FOUND</a>	51
<i>error code: more than one user matror the que y</i>	

<a href="#">patUSER TABLE DOES NOT EXIST</a>	52
<i>error code: table does not exist</i>	
<a href="#">patUSER USER ALREADY EXISTS</a>	52
<i>error code: user already exists</i>	
<a href="#">patUser::\$actionVar</a>	53
<i>name of the global variable that indicates the action in the request</i>	
<a href="#">patUser</a>	53
<i>patUser</i>	
<i>\$Id: patUser.php 2 2004-03-02 23:38:08Z schst \$</i>	
<a href="#">patUSER NO UNIQUE PRIMARY FOUND</a>	51
<i>error code: data matched more than one row</i>	
<a href="#">patUSER NO UNIQUE GROUP FOUND</a>	51
<i>error code: no unique group matched the query</i>	
<a href="#">patUSER NO DATA CHANGED</a>	50
<i>error code: no data was changed (affected rows = 0)</i>	
<a href="#">patUSER NOT IN GROUP</a>	50
<i>error code: user is not in group</i>	
<a href="#">patUSER NEED USERNAME</a>	49
<i>error code: function requires a user name</i>	
<a href="#">patUSER NO DATA GIVEN</a>	50
<i>error code: function requires data</i>	
<a href="#">patUSER NO DB RESULT</a>	50
<i>error code: query had no result</i>	
<a href="#">patUSER NO PRIMARY FOUND</a>	51
<i>error code: no primary key was found</i>	
<a href="#">patUSER NO GROUP FOUND</a>	51
<i>error code: no group matched the query</i>	
<a href="#">patUser::addTable()</a>	60
<i>add table that is used to store user data</i>	
<a href="#">patUser::addUser()</a>	61
<i>adds a user</i>	
<a href="#">patUser::removeLastError()</a>	75
<i>remove the last error from the list</i>	
<a href="#">patUser::modifyUser()</a>	75
<i>modify an existing user</i>	
<a href="#">patUser::modifyGroup()</a>	75
<i>modify an existing group</i>	
<a href="#">patUser::removeUserFromGroup()</a>	75
<i>delete user(s) from group(s)</i>	
<a href="#">patUser::requireAuthentication()</a>	76
<i>require an authenticated user</i>	
<a href="#">patUser::sendQuery()</a>	77
<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>	
<a href="#">patUser::searchUsers()</a>	76
<i>Search for users matching certain criterias</i>	
<a href="#">patUser::logout()</a>	74
<i>force log out of an authenticated user</i>	
<a href="#">patUser::keepHistory()</a>	74
<i>let patUser create a history of visited pages</i>	
<i>needs session support</i>	
<a href="#">patUser::hasPermission()</a>	72

<i>check, wether user has a permission</i>	72
<a href="#">patUser::goHistory()</a>	72
<i>go back x pages in the history as Header( "Location:..." ) is used, there must not be any output before this is called</i>	
<a href="#">patUser::getUsersInGroup()</a>	71
<i>get users in group</i>	
<a href="#">patUser::identifyGroup()</a>	73
<i>identify a group by certain fields</i>	
<a href="#">patUser::identifyUser()</a>	73
<i>identify a user by certain fields</i>	
<a href="#">patUser::isMemberOfGroup()</a>	73
<i>check, if a user is in a group</i>	
<a href="#">patUser::isAuthenticated()</a>	73
<i>check, wether a user is already authenticated if auth is done via session, all important vars will be fetched</i>	
<a href="#">patUser::setAuthDbc()</a>	77
<i>set dbc that contains auth info</i>	
<a href="#">patUser::setAuthFields()</a>	77
<i>set fieldnames that contain auth info</i>	
<a href="#">patUser::setTemplate()</a>	81
<i>set template object the template object is used for displaying login / logout screens</i>	
<a href="#">patUser::setRealm()</a>	81
<i>set authentication realm</i>	
<a href="#">patUser::setPermTable()</a>	81
<i>set tablename that contains permissions</i>	
<a href="#">patUser::setUnauthorizedURL()</a>	82
<i>set URL to redirect unauthorized users to</i>	
<a href="#">patUser::storeSessionValue()</a>	82
<i>store a value in the session may only be used if patUser was called with \$useSessions = true</i>	
<a href="#">patUser::updateStats()</a>	83
<i>update all statistics</i>	
<a href="#">patUser::translateErrorCode()</a>	82
<i>translate an error code</i>	
<a href="#">patUser::setPermFields()</a>	80
<i>set fieldnames that are used in the permissions</i>	
<a href="#">patUser::setMaxLoginAttempts()</a>	80
<i>set maximum amount of login attempts</i>	
<a href="#">patUser::setError()</a>	78
<i>set error code</i>	
<a href="#">patUser::setCryptFunction()</a>	78
<i>set function that is used for passwd encryption</i>	
<a href="#">patUser::setAuthTable()</a>	78
<i>set tablename that contains auth info</i>	
<a href="#">patUser::setGroupFields()</a>	79
<i>set fieldnames that contain group info</i>	
<a href="#">patUser::setGroupRelFields()</a>	79
<i>set fieldnames that are used in the user group relations</i>	
<a href="#">patUser::setGroupTable()</a>	80
<i>set tablename that contains group info</i>	
<a href="#">patUser::setGroupRelTable()</a>	79

<i>set tablename that contains user group relations</i>	
<a href="#">patUser::getUsers()</a>	71
<i>get users from any table</i>	
<a href="#">patUser::getUserData()</a>	71
<i>fetch the user data from a table</i>	
<a href="#">patUser::getAllErrorCodes()</a>	65
<i>get all error codes</i>	
<a href="#">patUser::encryptPasswd()</a>	64
<i>encrypt a password using the specified encryption function</i>	
<a href="#">patUser::deleteUser()</a>	64
<i>delete (data of) an existing user</i>	
<a href="#">patUser::getAllErrorMessage()</a>	65
<i>get all error messages</i>	
<a href="#">patUser::getAllErrors()</a>	65
<i>get all error codes AND messages</i>	
<a href="#">patUser::getDbc()</a>	66
<i>get the dbc object for a table</i>	
<a href="#">patUser::getAuthFields()</a>	65
<i>get fieldnames that contain auth info</i>	
<a href="#">patUser::deletePermission()</a>	64
<i>delete permission(s)</i>	
<a href="#">patUser::deleteGroup()</a>	63
<i>delete an existing group</i>	
<a href="#">patUser::clearErrors()</a>	62
<i>clear all errors</i>	
<a href="#">patUser::authenticate()</a>	61
<i>authenticate a user</i>	
<a href="#">patUser::addUserToGroup()</a>	61
<i>add a user to a group</i>	
<a href="#">patUser::clearSessionValue()</a>	62
<i>clear a value in the session</i>	
<i>may only be used if patUser was called with \$useSessions = true</i>	
<a href="#">patUser::convertOptionsToSql()</a>	62
<i>convert options to sql</i>	
<a href="#">patUser::deleteAllPermissions()</a>	63
<i>delete all permissions without setting new ones</i>	
<a href="#">patUser::countTableEntries()</a>	63
<i>count amount of entries for a user in a table</i>	
<a href="#">patUser::getField()</a>	66
<i>fetch the value of a field</i>	
<a href="#">patUser::getGroupData()</a>	66
<i>fetch the group data from group table</i>	
<a href="#">patUser::getSelfUrl()</a>	69
<a href="#">patUser::getPrimaryValue()</a>	69
<i>get the value of the primary key in an additional table</i>	
<i>identifies a unique line in an additional table</i>	
<a href="#">patUser::getPermTable()</a>	69
<i>get tablename that contains permissions</i>	
<a href="#">patUser::getSessionValue()</a>	69
<i>get a value that was stored in the session</i>	
<i>may only be used if patUser was called with \$useSessions = true</i>	
<a href="#">patUser::getTableDef()</a>	70
<i>get table definitions</i>	

<a href="#">patUser::getUid()</a>	70
<i>return the user id of the current user</i>	
<a href="#">patUser::getTableInfo()</a>	70
<i>get information about table fields like DESCRIBE [table]</i>	
<a href="#">patUser::getPermissions()</a>	68
<i>get all permissions of a user</i>	
<a href="#">patUser::getLastErrorMessage()</a>	68
<i>get the last error message</i>	
<a href="#">patUser::getGroups()</a>	67
<i>get groups from the grouptable</i>	
<a href="#">patUser::getGroupFields()</a>	66
<i>get fieldnames that contain group info</i>	
<a href="#">patUser::getHistory()</a>	67
<i>get the history for the current user</i>	
<a href="#">patUser::getJoinedGroups()</a>	67
<i>get joined groups</i>	
<a href="#">patUser::getLastErrorCode()</a>	68
<i>get the last error code</i>	
<a href="#">patUser::getLastError()</a>	68
<i>get the last error code AND message</i>	
<a href="#">patMySqlResult::field_len()</a>	43
<i>returns the length of the specified field</i>	
<a href="#">patMySqlResult::fetch_row()</a>	43
<i>fetch one row of the result</i>	
<a href="#">patServer::\$domain</a>	18
<i>domain to bind to</i>	
<a href="#">patServer::\$debugMode</a>	18
<i>debug mode</i>	
<a href="#">patServer::\$debugDest</a>	18
<i>debug destination (filename or stdout)</i>	
<a href="#">patServer::\$maxClients</a>	18
<i>maximum amount of clients</i>	
<a href="#">patServer::\$maxQueue</a>	18
<i>maximum of backlog in queue</i>	
<a href="#">patServer::\$port</a>	19
<i>port to listen</i>	
<a href="#">patServer::\$null</a>	19
<i>empty array, used for socket_select</i>	
<a href="#">patServer::\$debug</a>	18
<i>debug mode</i>	
<a href="#">patServer::\$clients</a>	18
<i>amount of clients</i>	
<a href="#">patPaintServer</a>	16
<i>patServer</i>	
<a href="#">patJabberServer::startElement()</a>	16
<a href="#">patJabberServer::endElement()</a>	16
<a href="#">patPaintServer::onReceiveData()</a>	17
<a href="#">patServer</a>	17
<i>patServer</i>	
<a href="#">patServer::\$clientInfo</a>	18
<i>needed to store client information</i>	
<a href="#">patServer::\$clientFD</a>	17

<i>all file descriptors are stored here</i>	
<a href="#">patServer::\$readBufferSize</a>	19
<i>buffer size for socket_read</i>	
<a href="#">patServer::\$readEndCharacter</a>	19
<i>end character for socket_read</i>	
<a href="#">patServer::start()</a>	23
<i>start the server</i>	
<a href="#">patServer::shutDown()</a>	23
<i>shutdown server</i>	
<a href="#">patServer::setMaxClients()</a>	23
<i>set maximum amount of simultaneous connections</i>	
<a href="#">patTelnetChatServer</a>	24
<i>patTelnetChatServer</i>	
<i>simple example to demonstrate how patServer is used</i>	
<a href="#">patTelnetChatServer::\$users</a>	24
<a href="#">patTicTacToe</a>	24
<i>patServer</i>	
<a href="#">patTelnetChatServer::getClientIdByNick()</a>	24
<i>get id of file descriptor by nickname</i>	
<a href="#">patServer::setDebugMode()</a>	22
<i>set debug mode</i>	
<a href="#">patServer::sendData()</a>	22
<i>send data to a client</i>	
<a href="#">patServer::closeConnection()</a>	20
<i>close connection to a client</i>	
<a href="#">patServer::broadcastData()</a>	20
<i>send data to all clients</i>	
<a href="#">patServer::acceptConnection()</a>	19
<i>accept a new connection</i>	
<a href="#">patServer::getClientInfo()</a>	21
<i>get current information about a client</i>	
<a href="#">patServer::getClients()</a>	21
<i>get current amount of clients</i>	
<a href="#">patServer::isConnected()</a>	21
<i>check, whether a client is still connected</i>	
<a href="#">patServer::getLastSocketError()</a>	21
<i>return string for last socket error</i>	
<a href="#">patJabberServer::destroyParser()</a>	16
<a href="#">patJabberServer::cData()</a>	16
<a href="#">patFlashChatServer::\$users</a>	11
<i>data for online users</i>	
<a href="#">patFlashChatServer::\$user</a>	10
<i>patUser object</i>	
<a href="#">patFlashChatServer::\$readEndCharacter</a>	10
<i>end character for socket_read</i>	
<a href="#">patFlashChatServer::doLogin()</a>	11
<a href="#">patFlashChatServer::onClose()</a>	11
<a href="#">patFlashChatServer::processTextMessage()</a>	12
<a href="#">patFlashChatServer::onReceiveRequest()</a>	11
<i>request by client has been received</i>	
<a href="#">patFlashChatServer::\$dbc</a>	10
<i>patDbc object</i>	
<a href="#">patFlashChatServer</a>	10

<i>patFlashChatServer</i>	
<a href="#">patChatServer::onConnect()</a>	9
<a href="#">patChatServer::onClose()</a>	9
<a href="#">patChatServer</a>	9
<i>patServer</i>	
<a href="#">patChatServer::onConnectionRefused()</a>	9
<a href="#">patChatServer::onReceiveData()</a>	9
<a href="#">patChatServer::onStart()</a>	10
<a href="#">patChatServer::onShutdown()</a>	10
<a href="#">patFlashChatServer::quit()</a>	12
<i>quit</i>	
<a href="#">patHTTPServer</a>	12
<i>patHTTPServer</i>	
<i>simple example to demonstrate how patServer is used</i>	
<i>Do NOT use in production environments, as it's still quite buggy!</i>	
<a href="#">patJabberServer::\$inStream</a>	15
<i>flag to indicate whether a connection is in stream mode</i>	
<a href="#">patJabberServer::\$inMessage</a>	15
<i>flag to indicate whether a connection is in a message</i>	
<a href="#">patJabberServer::\$attStack</a>	15
<i>stack for attributes</i>	
<a href="#">patJabberServer::\$readBufferSize</a>	15
<i>buffer size for socket_read</i>	
<a href="#">patJabberServer::\$tagStack</a>	15
<i>stack for tags</i>	
<a href="#">patJabberServer::onClose()</a>	16
<i>client closes the connection</i>	
<a href="#">patJabberServer::\$xmlParser</a>	15
<i>xml parsers for all clients</i>	
<a href="#">patJabberServer</a>	14
<i>patServer</i>	
<a href="#">patHTTPServer::stripTrailingSlash()</a>	14
<i>strip trailing slash from a path</i>	
<a href="#">patHTTPServer::getContentType()</a>	13
<i>get content type of a file</i>	
<i>content types are defined in the httpd.conf file</i>	
<a href="#">patHTTPServer::\$readEndCharacter</a>	12
<i>end character for socket_read</i>	
<a href="#">patHTTPServer::onReceiveData()</a>	13
<i>data was received, i.e. HTTP request sent</i>	
<a href="#">patHTTPServer::onStart()</a>	13
<i>server is started</i>	
<a href="#">patHTTPServer::parseRequest()</a>	14
<i>parse a http request</i>	
<a href="#">patHTTPServer::parsePath()</a>	13
<i>parse a request uri</i>	
<a href="#">patTicTacToe::\$candidates</a>	25
<i>win candidates</i>	
<a href="#">patTicTacToe::\$emptyField</a>	25
<i>template for a new game</i>	
<a href="#">patDbcResult::num_fields()</a>	37
<i>get the number of fields in the result</i>	
<a href="#">patDbcResult::get_result()</a>	37

<a href="#">get the complete result as two dimensional array</a>	36
<a href="#">patDbcResult::free()</a>	36
<i>free used memory used by the result</i>	
<a href="#">patDbcResult::num_rows()</a>	37
<i>get the number of rows in the result</i>	
<a href="#">patDbcResult::setIdentifier()</a>	37
<i>set the result identifier</i>	
<a href="#">patMySqlDbc::\$dbhost</a>	38
<a href="#">patMySqlDbc</a>	38
<i>MySql Abstraction</i>	
<i>provides commonly needed mysql functions</i>	
<a href="#">patDbcResult::fetch_row()</a>	36
<i>fetch one row of the result</i>	
<a href="#">patDbcResult::fetch_array()</a>	36
<i>fetch one row of the result as associative array</i>	
<a href="#">patDBC_TYPENUM</a>	34
<a href="#">patDBC_TYPEBOTH</a>	34
<a href="#">patDBC_TYPEASSOC</a>	34
<a href="#">patDbcResult</a>	35
<i>generic result object, will be extended by result objects for databases</i>	
<a href="#">patDbcResult::\$id</a>	35
<a href="#">patDbcResult::dumpRow()</a>	36
<i>dump one row</i>	
<a href="#">patDbcResult::dump()</a>	35
<i>dump the result</i>	
<a href="#">patMySqlDbc::\$dbid</a>	38
<a href="#">patMySqlDbc::\$dbname</a>	38
<a href="#">patMySqlResult</a>	41
<i>mySql result object</i>	
<i>result object returned by the query() method of the patMySqlDbc</i>	
<a href="#">patMySqlDbc::query()</a>	41
<i>Send a query to mysql</i>	
<a href="#">patMySqlDbc::insert_id()</a>	41
<i>get the last insert id for this connection</i>	
<a href="#">patMySqlResult::data_seek()</a>	41
<i>move the internal row pointer of the result</i>	
<a href="#">patMySqlResult::fetch_array()</a>	42
<i>fetch one row of the result as associative array</i>	
<a href="#">patMySqlResult::fetch_object()</a>	43
<i>fetch one row of the result as an object</i>	
<a href="#">patMySqlResult::fetch_field()</a>	42
<i>fetch information about a field of the result</i>	
<a href="#">patMySqlDbc::get_fields()</a>	40
<i>get fieldnames of a mysql table</i>	
<a href="#">patMySqlDbc::drop_db()</a>	40
<i>drop a database</i>	
<a href="#">patMySqlDbc::\$status</a>	38
<a href="#">patMySqlDbc::\$dbuser</a>	38
<a href="#">patMySqlDbc::\$dbpass</a>	38
<a href="#">patMySqlDbc::affected_rows()</a>	39
<i>count the number of affected rows</i>	
<a href="#">patMySqlDbc::close()</a>	39
<i>Close the MySql connection</i>	

<a href="#">patMySqlDbc::create_db()</a>	39
<i>create a database</i>	
<a href="#">patMySqlDbc::connect()</a>	39
<i>Open the MySql connection</i>	
<a href="#">patDBC_OPEN</a>	34
<a href="#">patDBC_CLOSED</a>	34
<a href="#">patTicTacToe::onReceiveRequest()</a>	27
<i>request by flash received</i>	
<a href="#">patTicTacToe::onClose()</a>	27
<i>client closed connection</i>	
<a href="#">patTicTacToe::getOpponentId()</a>	26
<i>get id of the opponent a player</i>	
<a href="#">patTicTacToe::putOnHold()</a>	27
<i>put player on hold</i>	
<a href="#">patTicTacToe::sendResponseToPlayers()</a>	27
<i>send a response to both players of a game</i>	
<a href="#">patXMLChatServer</a>	28
<i>patServer</i>	
<a href="#">patTicTacToe::startGame()</a>	28
<i>start a game</i>	
<a href="#">patTicTacToe::getGameId()</a>	26
<i>get id of a game for a player</i>	
<a href="#">patTicTacToe::endGame()</a>	26
<i>end a game</i>	
<a href="#">patTicTacToe::\$players</a>	25
<i>all players</i>	
<a href="#">patTicTacToe::\$games</a>	25
<i>games that are in use</i>	
<a href="#">patTicTacToe::\$gameId</a>	25
<i>id of the next game</i>	
<a href="#">patTicTacToe::\$readEndCharacter</a>	26
<i>end character for socket_read</i>	
<a href="#">patTicTacToe::\$waitingPlayers</a>	26
<i>users that are waiting to play (normally only one)</i>	
<a href="#">patTicTacToe::createNewGame()</a>	26
<i>create a new game</i>	
<a href="#">patTicTacToe::checkForNewGames()</a>	26
<a href="#">patXMLChatServer::\$users</a>	28
<a href="#">patXMLChatServer::doLogin()</a>	28
<a href="#">patXMLServer_Dom</a>	30
<i>patServer</i>	
<a href="#">patXMLServer::startElement()</a>	30
<a href="#">patXMLServer::parseXML()</a>	30
<a href="#">patXMLServer_Dom::broadcastResponse()</a>	31
<i>send response to all clients</i>	
<a href="#">patXMLServer_Dom::encodeResponse()</a>	31
<i>encode a request</i>	
<a href="#">patDbc.php</a>	34
<a href="#">patXMLServer_Dom::sendResponse()</a>	31
<i>send a response</i>	
<a href="#">patXMLServer::onStart()</a>	30
<a href="#">patXMLServer::endElement()</a>	30
<a href="#">patXMLChatServer::onReceiveData()</a>	29

<a href="#">patXMLChatServer::onClose()</a>	28
<a href="#">patXMLServer</a>	29
<i>patServer</i>	
<a href="#">patXMLServer::\$parser</a>	29
<a href="#">patXMLServer::characterData()</a>	29
<a href="#">patXMLServer::buildXML()</a>	29
<a href="#">patTTT.php</a>	5

## T

<a href="#">telnetchat_pear.php</a>	7
<a href="#">telnetchat.php</a>	6