Memcache

Introduction

Memcache module provides handy procedural and object oriented interface to memcached, highly effective caching daemon, which was especially designed to decrease database load in dynamic web applications.

The Memcache module also provides a session handler (*memcache*).

More information about memcached can be found at
» http://www.danga.com/memcached/.

Installing/Configuring

Requirements

This module uses functions of » zlib to support on-the-fly data compression. Zlib is required to install this module.

PHP 4.3.3 or newer is required to use the memcache extension.

Installation

This » PECL extension is not bundled with PHP. Information for installing this PECL extension may be found in the manual chapter titled Installation of PECL extensions. Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: » http://pecl.php.net/package/memcache.

In order to use these functions you must compile PHP with Memcache support by using the *--enable-memcache[=DIR]* option. You may optionally disable memcache session handler support by specifying *--disable-memcache-session*.

Windows users will enable *php_memcache.dll* inside of *php.ini* in order to use these functions. The DLL for this PECL extension may be downloaded from either the » PHP Downloads page or from » http://pecl4win.php.net/

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Memcache Configuration Options

Name	Default	Changeable	Changelog
memcache.allow_failover	"1"	PHP_INI_ALL	Available since memcache 2.0.2.
memcache.max_failover_attempts	"20"	PHP_INI_ALL	Available since memcache 2.1.0.
memcache.chunk_size	"8192"	PHP_INI_ALL	Available since memcache 2.0.2.
memcache.default_port	"11211"	PHP_INI_ALL	Available since memcache 2.0.2.

memcache.hash_strategy	"standard"	PHP_INI_ALL	Available since memcache 2.2.0.
memcache.hash_function	"crc32"	PHP_INI_ALL	Available since memcache 2.2.0.
session.save_handler	"files"	PHP_INI_ALL	Supported since memcache 2.1.2
session.save_path	""	PHP_INI_ALL	Supported since memcache 2.1.2

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

memcache.allow_failover boolean
: Whether to transparently failover to other servers on errors.

memcache.max_failover_attempts integer
: Defines how many servers to try when setting and getting data. Used only in conjunction with memcache.allow_failover.

memcache.chunk_size integer
: Data will be transferred in chunks of this size, setting the value lower requires more network writes. Try increasing this value to 32768 if noticing otherwise inexplicable slowdowns.

memcache.default_port string
: The default TCP port number to use when connecting to the memcached server if no other port is specified.

memcache.hash_strategy string
: Controls which strategy to use when mapping keys to servers. Set this value to *consistent* to enable consistent hashing which allows servers to be added or removed from the pool without causing keys to be remapped. Setting this value to *standard* results in the old strategy being used.

memcache.hash_function string
: Controls which hash function to apply when mapping keys to servers, *crc32* uses the standard CRC32 hash while *fnv* uses FNV-1a.

session.save_handler string
: Use memcache as a session handler by setting this value to *memcache*.

session.save_path string
: Defines a comma separated of server urls to use for session storage, for example *"tcp://host1:11211, tcp://host2:11211"*. Each url may contain parameters which are applied to that server, they are the same as for the Memcache::addServer() method. For example *"tcp://host1:11211?persistent=1&weight=1&timeout=1&retry_interval=15"*

Resource Types

There is only one resource type used in memcache module - it's the link identifier for a cache server connection.

Predefined Constants

Memcache Constants

Name	Description
MEMCACHE_COMPRESSED (integer)	Used to turn on-the-fly data compression on with Memcache::set(), Memcache::add() and Memcache::replace().
MEMCACHE_HAVE_SESSION (integer)	1 if this Memcache session handler is available, 0 otherwise.

Examples

Example #1 - memcache extension overview example

In this example, an object is being saved in the cache and then retrieved back. Object and other non-scalar types are serialized before saving, so it's impossible to store resources (i.e. connection identifiers and others) in the cache.

```
<?php

$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die ("Could not connect");

$version = $memcache->getVersion();
echo "Server's version: ".$version."<br/>\n";

$tmp_object = new stdClass;
$tmp_object->str_attr = 'test';
$tmp_object->int_attr = 123;

$memcache->set('key', $tmp_object, false, 10) or die ("Failed to save data
at the server");
echo "Store data in the cache (data will expire in 10 seconds)<br/>\n";

$get_result = $memcache->get('key');
echo "Data from the cache:<br/>\n";

var_dump($get_result);

?>
```

Example #2 - Using memcache session handler

```
<?php

$session_save_path =
"tcp://$host:$port?persistent=1&weight=2&timeout=2&retry_interval=10,
,tcp://$host:$port  ";
ini_set('session.save_handler', 'memcache');
ini_set('session.save_path', $session_save_path);

?>
```

Memcache Functions

Memcache::add

Memcache::add -- Add an item to the server

Description

bool **Memcache::add** (string `$key`, mixed `$var` [, int `$flag` [, int `$expire`]])

Memcache::add() stores variable *`var`* with *`key`* only if such key doesn't exist at the server yet. Also you can use memcache_add() function.

Parameters

`key`
: The key that will be associated with the item.

`var`
: The variable to store. Strings and integers are stored as is, other types are stored serialized.

`flag`
: Use **MEMCACHE_COMPRESSED** to store the item compressed (uses zlib).

`expire`
: Expiration time of the item. If it's equal to zero, the item will never expire. You can also use Unix timestamp or a number of seconds starting from current time, but in the latter case the number of seconds may not exceed 2592000 (30 days).

Return Values

Returns **TRUE** on success or **FALSE** on failure. Returns **FALSE** if such key already exist. For the rest Memcache::add() behaves similarly to Memcache::set().

Examples

Example #3 - Memcache::add() example

```
<?php

$memcache_obj = memcache_connect("localhost", 11211);

/* procedural API */
memcache_add($memcache_obj, 'var_key', 'test variable', false, 30);

/* OO API */
$memcache_obj->add('var_key', 'test variable', false, 30);
```

```
?>
```

See Also

- Memcache::set()
- Memcache::replace()

Memcache::addServer

Memcache::addServer -- Add a memcached server to connection pool

Description

bool **Memcache::addServer** (string `$host` [, int `$port` [, bool `$persistent` [, int `$weight` [, int `$timeout` [, int `$retry_interval` [, bool `$status` [, callback `$failure_callback`]]]]]]])

Memcache::addServer() adds a server to the connection pool. The connection, which was opened using Memcache::addServer() will be automatically closed at the end of script execution, you can also close it manually with Memcache::close(). You can also use the **memcache_add_server()** function.

When using this method (as opposed to **Memcache::connect()()** and **Memcache::pconnect()()**) the network connection is not established until actually needed. Thus there is no overhead in adding a large number of servers to the pool, even though they might not all be used.

Failover may occur at any stage in any of the methods, as long as other servers are available the request the user won't notice. Any kind of socket or Memcached server level errors (except out-of-memory) may trigger the failover. Normal client errors such as adding an existing key will not trigger a failover.

Note
This function has been added to Memcache version 2.0.0.

Parameters

`host`
: Point to the host where memcached is listening for connections. This parameter may also specify other transports like *unix:///path/to/memcached.sock* to use UNIX domain sockets, in this case *`port`* must also be set to *0*.

`port`
: Point to the port where memcached is listening for connections. This parameter is optional and its default value is 11211. Set this parameter to *0* when using UNIX domain sockets.

`persistent`
: Controls the use of a persistent connection. Default to **TRUE**.

`weight`
: Number of buckets to create for this server which in turn control its probability of it

being selected. The probability is relative to the total weight of all servers.

timeout
: Value in seconds which will be used for connecting to the daemon. Think twice before changing the default value of 1 second - you can lose all the advantages of caching if your connection is too slow.

retry_interval
: Controls how often a failed server will be retried, the default value is 15 seconds. Setting this parameter to -1 disables automatic retry. Neither this nor the *persistent* parameter has any effect when the extension is loaded dynamically via dl(). Each failed connection struct has its own timeout and before it has expired the struct will be skipped when selecting backends to serve a request. Once expired the connection will be successfully reconnected or marked as failed for another *retry_interval* seconds. The typical effect is that each web server child will retry the connection about every *retry_interval* seconds when serving a page.

status
: Controls if the server should be flagged as online. Setting this parameter to **FALSE** and *retry_interval* to -1 allows a failed server to be kept in the pool so as not to affect the key distribution algoritm. Requests for this server will then failover or fail immediatly depending on the *memcache.allow_failover* setting. Default to **TRUE**, meaning the server should be considered online.

failure_callback
: Allows the user to specify a callback function to run upon encountering an error. The callback is run before failover is attemped. The function takes two parameters, the hostname and port of the failed server.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #4 - Memcache::addServer() example

```
<?php

/* OO API */

$memcache = new Memcache;
$memcache->addServer('memcache_host', 11211);
$memcache->addServer('memcache_host2', 11211);

/* procedural API */

$memcache_obj = memcache_connect('memcache_host', 11211);
memcache_add_server($memcache_obj, 'memcache_host2', 11211);
```

```
?>
```

See Also

- Memcache::connect()
- Memcache::pconnect()
- Memcache::close()
- Memcache::setServerParams()
- Memcache::getServerStatus()

Memcache::close

Memcache::close -- Close memcached server connection

Description

bool **Memcache::close** (void)

Memcache::close() closes connection to memcached server. This function doesn't close persistent connections, which are closed only during web-server shutdown/restart. Also you can use memcache_close() function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #5 - Memcache::close() example

```
<?php

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);
/*
do something here ..
*/
memcache_close($memcache_obj);

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);
/*
do something here ..
*/
$memcache_obj->close();

?>
```

See Also

- Memcache::connect()
- Memcache::pconnect()

Memcache::connect

Memcache::connect -- Open memcached server connection

Description

bool **Memcache::connect** (string `$host` [, int `$port` [, int `$timeout`]])

Memcache::connect() establishes a connection to the memcached server. The connection, which was opened using Memcache::connect() will be automatically closed at the end of script execution. Also you can close it with Memcache::close(). Also you can use memcache_connect() function.

Parameters

`host`
: Point to the host where memcached is listening for connections. This parameter may also specify other transports like *unix:///path/to/memcached.sock* to use UNIX domain sockets, in this case *`port`* must also be set to *0*.

`port`
: Point to the port where memcached is listening for connections. Set this parameter to *0* when using UNIX domain sockets.

`timeout`
: Value in seconds which will be used for connecting to the daemon. Think twice before changing the default value of 1 second - you can lose all the advantages of caching if your connection is too slow.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #6 - Memcache::connect() example

```
<?php

/* procedural API */

$memcache_obj = memcache_connect('memcache_host', 11211);

/* OO API */

$memcache = new Memcache;
```

```
$memcache->connect('memcache_host', 11211);

?>
```

See Also

- Memcache::pconnect()
- Memcache::close()

memcache_debug

memcache_debug -- Turn debug output on/off

Description

bool **memcache_debug** (bool `$on_off`)

memcache_debug() turns on debug output if parameter *`on_off`* is equal to **TRUE** and turns off if it's **FALSE**.

Note
memcache_debug() is accessible only if PHP was built with --enable-debug option and always returns **TRUE** in this case. Otherwise, this function has no effect and always returns **FALSE**.

Parameters

`on_off`
: Turns debug output on if equals to **TRUE**. Turns debug output off if equals to **FALSE**.

Return Values

Returns **TRUE** if PHP was built with --enable-debug option, otherwise returns **FALSE**.

Memcache::decrement

Memcache::decrement -- Decrement item's value

Description

int **Memcache::decrement** (string `$key` [, int `$value`])

Memcache::decrement() decrements value of the item by *`value`*. Similarly to Memcache::increment(), current value of the item is being converted to numerical and after that *`value`* is substracted.

Note
New item's value will not be less than zero.

Note
Do not use Memcache::decrement() with item, which was stored compressed, because consequent call to Memcache::get() will fail.

Memcache::decrement() *does not* create an item if it didn't exist. Also you can use memcache_decrement() function.

Parameters

`key`
: Key of the item do decrement.

`value`
: Decrement the item by *`value`*. Optional and defaults to 1.

Return Values

Returns item's new value on success or **FALSE** on failure.

Examples

Example #7 - Memcache::decrement() example

```
<?php

/* procedural API */
```

```
$memcache_obj = memcache_connect('memcache_host', 11211);
/* decrement item by 2 */
$new_value = memcache_decrement($memcache_obj, 'test_item', 2);

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);
/* decrement item by 3 */
$new_value = $memcache_obj->decrement('test_item', 3);
?>
```

See Also

- Memcache::increment()
- Memcache::replace()

Memcache::delete

Memcache::delete -- Delete item from the server

Description

bool **Memcache::delete** (string `$key` [, int `$timeout`])

Memcache::delete() deletes item with the *`key`*. If parameter *`timeout`* is specified, the item will expire after *`timeout`* seconds. Also you can use memcache_delete() function.

Parameters

`key`
: The key associated with the item to delete.

`timeout`
: Execution time of the item. If it's equal to zero, the item will be deleted right away whereas if you set it to 30, the item will be deleted in 30 seconds.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #8 - Memcache::delete() example

```
<?php

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);

/* after 10 seconds item will be deleted by the server */
memcache_delete($memcache_obj, 'key_to_delete', 10);

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);

$memcache_obj->delete('key_to_delete', 10);

?>
```

See Also

- Memcache::set()
- Memcache::replace()

Memcache::flush

Memcache::flush -- Flush all existing items at the server

Description

bool **Memcache::flush** (void)

Memcache::flush() immediately invalidates all existing items. Memcache::flush() doesn't actually free any resources, it only marks all the items as expired, so occupied memory will be overwritten by new items. Also you can use memcache_flush() function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #9 - Memcache::flush() example

```
<?php

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);

memcache_flush($memcache_obj);

/* OO API */

$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);

$memcache_obj->flush();

?>
```

Memcache::get

Memcache::get -- Retrieve item from the server

Description

string **Memcache::get** (string `$key` [, int `&$flags`])

array **Memcache::get** (array `$keys` [, array `&$flags`])

Memcache::get() returns previously stored data if an item with such *key* exists on the server at this moment.

You can pass array of keys to Memcache::get() to get array of values. The result array will contain only found key-value pairs.

Parameters

key
: The key or array of keys to fetch.

flags
: If present, flags fetched along with the values will be written to this parameter. These flags are the same as the ones given to for example Memcache::set(). The lowest byte of the int is reserved for pecl/memcache internal usage (e.g. to indicate compression and serialization status).

Return Values

Returns the string associated with the *key* or **FALSE** on failure or if such *key* was not found.

Examples

Example #10 - Memcache::get() example

```
<?php

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);
$var = memcache_get($memcache_obj, 'some_key');

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);
$var = $memcache_obj->get('some_key');
```

```
/*
You also can use array of keys as a parameter.
If such item wasn't found at the server, the result
array simply will not include such key.
*/

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);
$var = memcache_get($memcache_obj, Array('some_key', 'another_key'));

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);
$var = $memcache_obj->get(Array('some_key', 'second_key'));

?>
```

Memcache::getExtendedStats

Memcache::getExtendedStats -- Get statistics from all servers in pool

Description

array **Memcache::getExtendedStats** ([string `$type` [, int `$slabid` [, int `$limit`]]])

Memcache::getExtendedStats() returns a two-dimensional associative array with server statistics. Array keys correspond to host:port of server and values contain the individual server statistics. A failed server will have its corresponding entry set to **FALSE**. You can also use the **memcache_get_extended_stats()** function.

Note
This function has been added to Memcache version 2.0.0.

Parameters

`type`
: The type of statistics to fetch. Valid values are {reset, malloc, maps, cachedump, slabs, items, sizes}. According to the memcached protocol spec these additional arguments "are subject to change for the convenience of memcache developers".

`slabid`
: Used in conjunction with *`type`* set to cachedump to identify the slab to dump from. The cachedump command ties up the server and is strictly to be used for debugging purposes.

`limit`
: Used in conjunction with *`type`* set to cachedump to limit the number of entries to dump. Default value is 100.

Return Values

Returns a two-dimensional associative array of server statistics or **FALSE** on failure.

Examples

Example #11 - Memcache::getExtendedStats() **example**

```
<?php
   $memcache_obj = new Memcache;
```

```
   $memcache_obj->addServer('memcache_host', 11211);
   $memcache_obj->addServer('failed_host', 11211);

   $stats = $memcache_obj->getExtendedStats();
   print_r($stats);
?>
```

The above example will output:

```
Array
(
   [memcache_host:11211] => Array
       (
           [pid] => 3756
           [uptime] => 603011
           [time] => 1133810435
           [version] => 1.1.12
           [rusage_user] => 0.451931
           [rusage_system] => 0.634903
           [curr_items] => 2483
           [total_items] => 3079
           [bytes] => 2718136
           [curr_connections] => 2
           [total_connections] => 807
           [connection_structures] => 13
           [cmd_get] => 9748
           [cmd_set] => 3096
           [get_hits] => 5976
           [get_misses] => 3772
           [bytes_read] => 3448968
           [bytes_written] => 2318883
           [limit_maxbytes] => 33554432
       )

   [failed_host:11211] => false
)
```

See Also

- Memcache::getVersion()
- Memcache::getStats()

Memcache::getServerStatus

Memcache::getServerStatus -- Returns server status

Description

int **Memcache::getServerStatus** (string `$host` [, int `$port`])

Memcache::getServerStatus() returns a the servers online/offline status. You can also use **memcache_get_server_status()** function.

Note
This function has been added to Memcache version 2.1.0.

Parameters

`host`
: Point to the host where memcached is listening for connections.

`port`
: Point to the port where memcached is listening for connections. This parameter is optional and its default value is 11211.

Return Values

Returns a the servers status. 0 if server is failed, non-zero otherwise

Examples

Example #12 - Memcache::getServerStatus() example

```
<?php

/* OO API */
$memcache = new Memcache;
$memcache->addServer('memcache_host', 11211);
echo $memcache->getServerStatus('memcache_host', 11211);

/* procedural API */
$memcache = memcache_connect('memcache_host', 11211);
echo memcache_get_server_status($memcache, 'memcache_host', 11211);

?>
```

See Also

- Memcache::addServer()
- Memcache::setServerParams()

Memcache::getStats

Memcache::getStats -- Get statistics of the server

Description

array **Memcache::getStats** ([string `$type` [, int `$slabid` [, int `$limit`]]])

Memcache::getStats() returns an associative array with server's statistics. Array keys correspond to stats parameters and values to parameter's values. Also you can use **memcache_get_stats()** function.

Parameters

`type`
: The type of statistics to fetch. Valid values are {reset, malloc, maps, cachedump, slabs, items, sizes}. According to the memcached protocol spec these additional arguments "are subject to change for the convenience of memcache developers".

`slabid`
: Used in conjunction with *`type`* set to cachedump to identify the slab to dump from. The cachedump command ties up the server and is strictly to be used for debugging purposes.

`limit`
: Used in conjunction with *`type`* set to cachedump to limit the number of entries to dump. Default value is 100.

Return Values

Returns an associative array of server statistics or **FALSE** on failure.

See Also

- Memcache::getVersion()
- Memcache::getExtendedStats()

Memcache::getVersion

Memcache::getVersion -- Return version of the server

Description

string **Memcache::getVersion** (void)

Memcache::getVersion() returns a string with server's version number. Also you can use **memcache_get_version()** function.

Return Values

Returns a string of server version number or **FALSE** on failure.

Examples

Example #13 - Memcache::getVersion() **example**

```
<?php

/* OO API */
$memcache = new Memcache;
$memcache->connect('memcache_host', 11211);
echo $memcache->getVersion();

/* procedural API */
$memcache = memcache_connect('memcache_host', 11211);
echo memcache_get_version($memcache);

?>
```

See Also

- Memcache::getExtendedStats()
- Memcache::getStats()

Memcache::increment

Memcache::increment -- Increment item's value

Description

int **Memcache::increment** (string `$key` [, int `$value`])

Memcache::increment() increments value of the item on the specified *`value`*. If item with key *`key`* was not numeric and cannot be converted to number, it will change it's value to *`value`*. Memcache::increment() *does not* create an item if it didn't exist.

Note
Do not use Memcache::increment() with item, which was stored compressed, because consequent call to Memcache::get() will fail.

Also you can use memcache_increment() function.

Parameters

`key`
: Key of the item to increment.

`value`
: Increment the item by *`value`*. Optional and defaults to 1.

Return Values

Returns new item's value on success or **FALSE** on failure.

Examples

Example #14 - Memcache::increment() example

```
<?php

/* procedural API */
$memcache_obj = memcache_connect('memcache_host', 11211);
/* increment counter by 2 */
$current_value = memcache_increment($memcache_obj, 'counter', 2);

/* OO API */
$memcache_obj = new Memcache;
$memcache_obj->connect('memcache_host', 11211);
/* increment counter by 3 */
```

```
$current_value = $memcache_obj->increment('counter', 3);

?>
```

See Also

- Memcache::decrement()
- Memcache::replace()

Memcache::pconnect

Memcache::pconnect -- Open memcached server persistent connection

Description

bool **Memcache::pconnect** (string `$host` [, int `$port` [, int `$timeout`]])

Memcache::pconnect() is similar to Memcache::connect() with the difference, that the connection it establishes is persistent. This connection is not closed after the end of script execution and by Memcache::close() function. Also you can use memcache_pconnect() function.

Parameters

`host`
: Point to the host where memcached is listening for connections. This parameter may also specify other transports like *unix:///path/to/memcached.sock* to use UNIX domain sockets, in this case *`port`* must also be set to *0*.

`port`
: Point to the port where memcached is listening for connections. Set this parameter to *0* when using UNIX domain sockets.

`timeout`
: Value in seconds which will be used for connecting to the daemon. Think twice before changing the default value of 1 second - you can lose all the advantages of caching if your connection is too slow.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #15 - Memcache::pconnect() example

```
<?php

/* procedural API */
$memcache_obj = memcache_pconnect('memcache_host', 11211);

/* OO API */

$memcache_obj = new Memcache;
$memcache_obj->pconnect('memcache_host', 11211);
```

```
?>
```

See Also

- Memcache::connect()

Memcache::replace

Memcache::replace -- Replace value of the existing item

Description

bool **Memcache::replace** (string `$key`, mixed `$var` [, int `$flag` [, int `$expire`]])

Memcache::replace() should be used to replace value of existing item with *key*. In case if item with such key doesn't exists, Memcache::replace() returns **FALSE**. For the rest Memcache::replace() behaves similarly to Memcache::set(). Also you can use memcache_replace() function.

Parameters

key
: The key that will be associated with the item.

var
: The variable to store. Strings and integers are stored as is, other types are stored serialized.

flag
: Use **MEMCACHE_COMPRESSED** to store the item compressed (uses zlib).

expire
: Expiration time of the item. If it's equal to zero, the item will never expire. You can also use Unix timestamp or a number of seconds starting from current time, but in the latter case the number of seconds may not exceed 2592000 (30 days).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #16 - Memcache::replace() example

```
<?php

$memcache_obj = memcache_connect('memcache_host', 11211);

/* procedural API */
memcache_replace($memcache_obj, "test_key", "some variable", false, 30);

/* OO API */
```

```
$memcache_obj->replace("test_key", "some variable", false, 30);

?>
```

See Also

- Memcache::set()
- Memcache::add()

Memcache::set

Memcache::set -- Store data at the server

Description

bool **Memcache::set** (string `$key`, mixed `$var` [, int `$flag` [, int `$expire`]])

Memcache::set() stores an item *`var`* with *`key`* on the memcached server. Parameter *`expire`* is expiration time in seconds. If it's 0, the item never expires (but memcached server doesn't guarantee this item to be stored all the time, it could be deleted from the cache to make place for other items). You can use **MEMCACHE_COMPRESSED** constant as *`flag`* value if you want to use on-the-fly compression (uses zlib).

Note
Remember that resource variables (i.e. file and connection descriptors) cannot be stored in the cache, because they cannot be adequately represented in serialized state.

Also you can use memcache_set() function.

Parameters

`key`
: The key that will be associated with the item.

`var`
: The variable to store. Strings and integers are stored as is, other types are stored serialized.

`flag`
: Use **MEMCACHE_COMPRESSED** to store the item compressed (uses zlib).

`expire`
: Expiration time of the item. If it's equal to zero, the item will never expire. You can also use Unix timestamp or a number of seconds starting from current time, but in the latter case the number of seconds may not exceed 2592000 (30 days).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #17 - Memcache::set() example

```
<?php
/* procedural API */

/* connect to memcached server */
$memcache_obj = memcache_connect('memcache_host', 11211);

/*
set value of item with key 'var_key'
using 0 as flag value, compression is not used
expire time is 30 seconds
*/
memcache_set($memcache_obj, 'var_key', 'some variable', 0, 30);

echo memcache_get($memcache_obj, 'var_key');

?>
```

Example #18 - Memcache::set() example

```
<?php
/* OO API */

$memcache_obj = new Memcache;

/* connect to memcached server */
$memcache_obj->connect('memcache_host', 11211);

/*
set value of item with key 'var_key', using on-the-fly compression
expire time is 50 seconds
*/
$memcache_obj->set('var_key', 'some really big variable',
MEMCACHE_COMPRESSED, 50);

echo $memcache_obj->get('var_key');

?>
```

See Also

- Memcache::add()
- Memcache::replace()

Memcache::setCompressThreshold

Memcache::setCompressThreshold -- Enable automatic compression of large values

Description

bool **Memcache::setCompressThreshold** (int `$threshold` [, float `$min_savings`])

Memcache::setCompressThreshold() enables automatic compression of large values. You can also use the **memcache_set_compress_threshold()** function.

Note
This function has been added to Memcache version 2.0.0.

Parameters

`threshold`
: Controls the minimum value length before attempting to compress automatically.

`min_saving`
: Specifies the minimum amount of savings to actually store the value compressed. The supplied value must be between 0 and 1. Default value is 0.2 giving a minimum 20% compression savings.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #19 - Memcache::setCompressThreshold() example

```
<?php

/* OO API */

$memcache_obj = new Memcache;
$memcache_obj->addServer('memcache_host', 11211);
$memcache_obj->setCompressThreshold(20000, 0.2);

/* procedural API */

$memcache_obj = memcache_connect('memcache_host', 11211);
```

```
memcache_set_compress_threshold($memcache_obj, 20000, 0.2);

?>
```

Memcache::setServerParams

Memcache::setServerParams -- Changes server parameters and status at runtime

Description

bool **Memcache::setServerParams** (string `$host` [, int `$port` [, int `$timeout` [, int `$retry_interval` [, bool `$status` [, callback `$failure_callback`]]]]])

Memcache::setServerParams() changes server parameters at runtime. You can also use the **memcache_set_server_params()** function.

Note
This function has been added to Memcache version 2.1.0.

Parameters

`host`
: Point to the host where memcached is listening for connections.

`port`
: Point to the port where memcached is listening for connections. This parameter is optional and its default value is 11211.

`timeout`
: Value in seconds which will be used for connecting to the daemon. Think twice before changing the default value of 1 second - you can lose all the advantages of caching if your connection is too slow.

`retry_interval`
: Controls how often a failed server will be retried, the default value is 15 seconds. Setting this parameter to -1 disables automatic retry. Neither this nor the *`persistent`* parameter has any effect when the extension is loaded dynamically via dl().

`status`
: Controls if the server should be flagged as online. Setting this parameter to **FALSE** and *`retry_interval`* to -1 allows a failed server to be kept in the pool so as not to affect the key distribution algoritm. Requests for this server will then failover or fail immediatly depending on the *`memcache.allow_failover`* setting. Default to **TRUE**, meaning the server should be considered online.

`failure_callback`
: Allows the user to specify a callback function to run upon encountering an error. The callback is run before failover is attemped. The function takes two parameters, the hostname and port of the failed server.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #20 - Memcache::setServerParams() example

```
<?php

function _callback_memcache_failure($host, $port) {
   print "memcache '$host:$port' failed";
}

/* OO API */

$memcache = new Memcache;

// Add the server in offline mode
$memcache->addServer('memcache_host', 11211, false, 1, 1, -1, false);

// Bring the server back online
$memcache->setServerParams('memcache_host', 11211, 1, 15, true,
'_callback_memcache_failure');

/* procedural API */

$memcache_obj = memcache_connect('memcache_host', 11211);
memcache_set_server_params($memcache_obj, 'memcache_host', 11211, 1, 15,
true, '_callback_memcache_failure');

?>
```

See Also

- Memcache::addServer()
- Memcache::getServerStatus()