Miscellaneous Functions

Introduction

These functions were placed here because none of the other categories seemed to fit.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

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

Misc. Configuration Options

Name	Default	Changeable	Changelog
ignore_user_abort	"0"	PHP_INI_ALL	
highlight.string	"#DD0000"	PHP_INI_ALL	
highlight.comment	"#FF8000"	PHP_INI_ALL	
highlight.keyword	"#007700"	PHP_INI_ALL	
highlight.bg	"#FFFFFF"	PHP_INI_ALL	Removed in PHP 6.0.0.
highlight.default	"#0000BB"	PHP_INI_ALL	
highlight.html	"#000000"	PHP_INI_ALL	
browscap	NULL	PHP_INI_SYSTEM	

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

Here's a short explanation of the configuration directives.

ignore_user_abort boolean
: **FALSE** by default. If changed to **TRUE** scripts will not be terminated after a client has aborted their connection. See also ignore_user_abort().

highlight.bg string *highlight.comment* string *highlight.default* string *highlight.html* string *highlight.keyword* string *highlight.string* string
: Colors for Syntax Highlighting mode. Anything that's acceptable in <font color="??????"> would work.

browscap string
: Name (e.g.: *browscap.ini*) and location of browser capabilities file. See also get_browser().

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

CONNECTION_ABORTED (integer)

CONNECTION_NORMAL (integer)

CONNECTION_TIMEOUT (integer)

__COMPILER_HALT_OFFSET__ (integer)
Added in PHP 5.1.

Misc. Functions

connection_aborted

connection_aborted -- Check whether client disconnected

Description

int **connection_aborted** (void)

Checks whether the client disconnected.

Return Values

Returns 1 if client disconnected, 0 otherwise.

See Also

- connection_status()
- ignore_user_abort()
- Connection Handling for a complete description of connection handling in PHP.

connection_status

connection_status -- Returns connection status bitfield

Description

int **connection_status** (void)

Gets the connection status bitfield.

Return Values

Returns the connection status bitfield, which can be used against the *CONNECTION_XXX* constants to determine the connection status.

See Also

- connection_aborted()
- ignore_user_abort()
- Connection Handling for a complete description of connection handling in PHP.

connection_timeout

connection_timeout -- Check if the script timed out

Description

int **connection_timeout** (void)

Determines whether the script timed out.

Return Values

Returns 1 if the script timed out, 0 otherwise.

Notes

Warning
Deprecated This function is deprecated, and doesn't even exist anymore as of 4.0.5.

See Also

- connection_status()
- Connection Handling for a complete description of connection handling in PHP.

constant

constant -- Returns the value of a constant

Description

mixed **constant** (string `$name`)

Return the value of the constant indicated by *`name`*.

constant() is useful if you need to retrieve the value of a constant, but do not know its name. I.e. it is stored in a variable or returned by a function.

This function works also with class constants.

Parameters

`name`
The constant name.

Return Values

Returns the value of the constant, or **NULL** if the constant is not defined.

Examples

Example #1 - constant() example

```
<?php

define("MAXSIZE", 100);

echo MAXSIZE;
echo constant("MAXSIZE"); // same thing as the previous line


interface bar {
   const test = 'foobar!';
}

class foo {
   const test = 'foobar!';
}

$const = 'test';

var_dump(constant('bar::'. $const)); // string(7) "foobar!"
```

```
var_dump(constant('foo::'. $const)); // string(7) "foobar!"

?>
```

See Also

- define()
- defined()
- The section on Constants

define

define -- Defines a named constant

Description

bool **define** (string `$name`, mixed `$value` [, bool `$case_insensitive`])

Defines a named constant at runtime.

Parameters

`name`
: The name of the constant.

`value`
: The value of the constant; only scalar and null values are allowed. Scalar values are integer, float, string or boolean values.

`case_insensitive`
: If set to **TRUE**, the constant will be defined case-insensitive. The default behaviour is case-sensitive; i.e. *CONSTANT* and *Constant* represent different values.

Return Values

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

Examples

Example #2 - Defining Constants

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.

define("GREETING", "Hello you.", true);
echo GREETING; // outputs "Hello you."
echo Greeting; // outputs "Hello you."

?>
```

See Also

- defined()
- constant()
- The section on Constants

defined

defined -- Checks whether a given named constant exists

Description

bool **defined** (string `$name`)

Checks whether the given constant exists and is defined.

Note
If you want to see if a variable exists, use isset() as defined() only applies to constants. If you want to see if a function exists, use function_exists().

Parameters

name
The constant name.

Return Values

Returns **TRUE** if the named constant given by *name* has been defined, **FALSE** otherwise.

Examples

Example #3 - Checking Constants

```
<?php
/* Note the use of quotes, this is important.  This example is checking
* if the string 'CONSTANT' is the name of a constant named CONSTANT */
if (defined('CONSTANT')) {
   echo CONSTANT;
}
?>
```

See Also

- define()

- constant()
- get_defined_constants()
- function_exists()
- The section on Constants

die

die -- Equivalent to exit()

Description

This language construct is equivalent to exit().

eval

eval -- Evaluate a string as PHP code

Description

mixed **eval** (string `$code_str`)

Evaluates the string given in *`code_str`* as PHP code. Among other things, this can be useful for storing code in a database text field for later execution.

There are some factors to keep in mind when using eval(). Remember that the string passed must be valid PHP code, including things like terminating statements with a semicolon so the parser doesn't die on the line after the eval(), and properly escaping things in *`code_str`*. To mix HTML output and PHP code you can use a closing PHP tag to leave PHP mode.

Also remember that variables given values under eval() will retain these values in the main script afterwards.

Parameters

`code_str`
: The code string to be evaluated. *`code_str`* does not have to contain PHP Opening tags. A *return* statement will immediately terminate the evaluation of the string .

Return Values

eval() returns **NULL** unless *return* is called in the evaluated code, in which case the value passed to *return* is returned. If there is a parse error in the evaluated code, eval() returns **FALSE** and execution of the following code continues normally. It is not possible to catch a parse error in eval() using set_error_handler().

Examples

Example #4 - eval() example - simple text merge

```
<?php
$string = 'cup';
$name = 'coffee';
$str = 'This is a $string with my $name in it.';
echo $str. "\n";
eval("\$str = \"$str\";");
echo $str. "\n";
?>
```

The above example will output:

```
This is a $string with my $name in it.
This is a cup with my coffee in it.
```

Notes

Note
Because this is a language construct and not a function, it cannot be called using variable functions

Tip
As with anything that outputs its result directly to the browser, the output-control functions can be used to capture the output of this function, and save it in a string (for example).

Note
In case of a fatal error in the evaluated code, the whole script exits.

See Also

- call_user_func()

exit

exit -- Output a message and terminate the current script

Description

void **exit** ([string `$status`])

void **exit** (int `$status`)

Terminates execution of the script.

Parameters

`status`
: If *`status`* is a string, this function prints the *`status`* just before exiting. If *`status`* is an integer, that value will also be used as the exit status. Exit statuses should be in the range 0 to 254, the exit status 255 is reserved by PHP and shall not be used. The status 0 is used to terminate the program successfully.

Note
PHP >= 4.2.0 does NOT print the *`status`* if it is an integer.

Return Values

No value is returned.

Examples

Example #5 - exit() example

```
<?php

$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
   or exit("unable to open file ($filename)");

?>
```

Example #6 - exit() status example

```
<?php

//exit program normally
exit;
exit();
exit(0);

//exit with an error code
exit(1);
exit(0376); //octal

?>
```

Notes

Note

Because this is a language construct and not a function, it cannot be called using variable functions

Note

This language construct is equivalent to die().

See Also

- register_shutdown_function()

get_browser

get_browser -- Tells what the user's browser is capable of

Description

mixed **get_browser** ([string `$user_agent` [, bool `$return_array`]])

Attempts to determine the capabilities of the user's browser, by looking up the browser's information in the *browscap.ini* file.

Parameters

`user_agent`
The User Agent to be analyzed. By default, the value of HTTP User-Agent header is used; however, you can alter this (i.e., look up another browser's info) by passing this parameter. You can bypass this parameter with a **NULL** value.

`return_array`
If set to **TRUE**, this function will return an array instead of an object.

Return Values

The information is returned in an object or an array which will contain various data elements representing, for instance, the browser's major and minor version numbers and ID string; **TRUE** / **FALSE** values for features such as frames, JavaScript, and cookies; and so forth.

The *cookies* value simply means that the browser itself is capable of accepting cookies and does not mean the user has enabled the browser to accept cookies or not. The only way to test if cookies are accepted is to set one with setcookie(), reload, and check for the value.

ChangeLog

Version	Description
4.3.2	The optional parameter *`return_array`* was added.

Examples

Example #7 - Listing all information about the users browser

```
<?php
echo $_SERVER['HTTP_USER_AGENT'] . "\n\n";

$browser = get_browser(null, true);
print_r($browser);
?>
```

The above example will output something similar to:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040803
Firefox/0.9.3

Array
(
   [browser_name_regex] => ^mozilla/5\.0 (windows; .; windows nt 5\.1;
.*rv:.*) gecko/.* firefox/0\.9.*$
   [browser_name_pattern] => Mozilla/5.0 (Windows; ?; Windows NT 5.1; *rv:*)
Gecko/* Firefox/0.9*
   [parent] => Firefox 0.9
   [platform] => WinXP
   [browser] => Firefox
   [version] => 0.9
   [majorver] => 0
   [minorver] => 9
   [css] => 2
   [frames] => 1
   [iframes] => 1
   [tables] => 1
   [cookies] => 1
   [backgroundsounds] =>
   [vbscript] =>
   [javascript] => 1
   [javaapplets] => 1
   [activexcontrols] =>
   [cdf] =>
   [aol] =>
   [beta] => 1
   [win16] =>
   [crawler] =>
   [stripper] =>
   [wap] =>
   [netclr] =>
)
```

Notes

Note

In order for this to work, your browscap configuration setting in *php.ini* must point to the correct location of the *browscap.ini* file on your system.

browscap.ini is not bundled with PHP, but you may find an up-to-date » php_browscap.ini file here.

While *browscap.ini* contains information on many browsers, it relies on user updates to keep the database current. The format of the file is fairly self-explanatory.

__halt_compiler

__halt_compiler -- Halts the compiler execution

Description

void **__halt_compiler** (void)

Halts the execution of the compiler. This can be useful to embed data in PHP scripts, like the installation files.

Byte position of the data start can be determined by the **__COMPILER_HALT_OFFSET__** constant which is defined only if there is a __halt_compiler() presented in the file.

Return Values

No value is returned.

Examples

Example #8 - A __halt_compiler() example

```
<?php

// open this file
$fp = fopen(__FILE__, 'r');

// seek file pointer to data
fseek($fp, __COMPILER_HALT_OFFSET__);

// and output it
var_dump(stream_get_contents($fp));

// the end of the script execution
__halt_compiler();the installation data (eg. tar, gz, PHP, etc.)
```

Notes

Note

__halt_compiler() can only be used from the outermost scope.

highlight_file

highlight_file -- Syntax highlighting of a file

Description

mixed **highlight_file** (string `$filename` [, bool `$return`])

Prints out or returns a syntax highlighted version of the code contained in *`filename`* using the colors defined in the built-in syntax highlighter for PHP.

Many servers are configured to automatically highlight files with a *phps* extension. For example, *example.phps* when viewed will show the syntax highlighted source of the file. To enable this, add this line to the *httpd.conf*:

```
AddType application/x-httpd-php-source .phps
```

Parameters

`filename`
Path to the PHP file to be highlighted.

`return`
Set this parameter to **TRUE** to make this function return the highlighted code.

Return Values

If *`return`* is set to **TRUE**, returns the highlighted code as a string instead of printing it out. Otherwise, it will return **TRUE** on success, **FALSE** on failure.

ChangeLog

Version	Description
4.2.1	This function is now also affected by safe_mode and open_basedir.
4.2.0	The *`return`* parameter was added.

Notes

Caution
Care should be taken when using the highlight_file() function to make sure that you do not inadvertently reveal sensitive information such as passwords or any other type of information that might create a potential security risk.

Note
This function uses internal output buffering with this parameter so it can not be used inside an ob_start() callback function.

See Also

- highlight_string()

highlight_string

highlight_string -- Syntax highlighting of a string

Description

mixed **highlight_string** (string `$str` [, bool `$return`])

Outputs or returns a syntax highlighted version of the given PHP code using the colors defined in the built-in syntax highlighter for PHP.

Parameters

`str`
The PHP code to be highligthed. This should include the opening tag.

`return`
Set this parameter to **TRUE** to make this function return the highlighted code.

Return Values

If *`return`* is set to **TRUE**, returns the highlighted code as a string instead of printing it out. Otherwise, it will return **TRUE** on success, **FALSE** on failure.

ChangeLog

Version	Description
4.2.0	The *`return`* parameter was added.

Examples

Example #9 - highlight_string() example

```
<?php
highlight_string('<?php phpinfo(); ?>');
?>
```

The above example will output (in PHP 4):

```
<code><font color="#000000">
```

```
<font color="#0000BB">&lt;?php phpinfo</font><font color="#007700">();
</font><font color="#0000BB">?&gt;</font>
</font>
</code>
```

The above example will output (in PHP 5):

```
<code><span style="color: #000000">
<span style="color: #0000BB">&lt;?php phpinfo</span><span style="color:
#007700">(); </span><span style="color: #0000BB">?&gt;</span>
</span>
</code>
```

Notes

Note
This function uses internal output buffering with this parameter so it can not be used inside an ob_start() callback function.

See Also

- highlight_file()

ignore_user_abort

ignore_user_abort -- Set whether a client disconnect should abort script execution

Description

int **ignore_user_abort** ([bool `$setting`])

Sets whether a client disconnect should cause a script to be aborted.

Parameters

`setting`
: If not set, the function will only return the current setting.

Return Values

Returns the previous setting, as a boolean.

Notes

PHP will not detect that the user has aborted the connection until an attempt is made to send information to the client. Simply using an echo statement does not guarantee that information is sent, see flush().

See Also

- connection_aborted()
- connection_status()
- Connection Handling for a complete description of connection handling in PHP.

pack

pack -- Pack data into binary string

Description

string **pack** (string `$format` [, mixed `$args` [, mixed `$...`]])

Pack given arguments into binary string according to *`format`*.

The idea for this function was taken from Perl and all formatting codes work the same as in Perl However, there are some formatting codes that are missing such as Perl's "u" format code.

Note that the distinction between signed and unsigned values only affects the function unpack(), where as function pack() gives the same result for signed and unsigned format codes.

Also note that PHP internally stores integer values as signed values of a machine-dependent size. If you give it an unsigned integer value too large to be stored that way it is converted to a float which often yields an undesired result.

Parameters

`format`

The *`format`* string consists of format codes followed by an optional repeater argument. The repeater argument can be either an integer value or * for repeating to the end of the input data. For a, A, h, H the repeat count specifies how many characters of one data argument are taken, for @ it is the absolute position where to put the next data, for everything else the repeat count specifies how many data arguments are consumed and packed into the resulting binary string. Currently implemented formats are:

pack() **format characters**

Code	Description
a	NUL-padded string
A	SPACE-padded string
h	Hex string, low nibble first
H	Hex string, high nibble first
c	signed char
C	unsigned char

s	signed short (always 16 bit, machine byte order)
S	unsigned short (always 16 bit, machine byte order)
n	unsigned short (always 16 bit, big endian byte order)
v	unsigned short (always 16 bit, little endian byte order)
i	signed integer (machine dependent size and byte order)
I	unsigned integer (machine dependent size and byte order)
l	signed long (always 32 bit, machine byte order)
L	unsigned long (always 32 bit, machine byte order)
N	unsigned long (always 32 bit, big endian byte order)
V	unsigned long (always 32 bit, little endian byte order)
f	float (machine dependent size and representation)
d	double (machine dependent size and representation)
x	NUL byte
X	Back up one byte
@	NUL-fill to absolute position

args

Return Values

Returns a binary string containing data.

Examples

Example #10 - pack() example

```
<?php
$binarydata = pack("nvc*", 0x1234, 0x5678, 65, 66);
?>
```

The resulting binary string will be 6 bytes long and contain the byte sequence 0x12, 0x34, 0x78, 0x56, 0x41, 0x42.

See Also

- unpack()

php_check_syntax

php_check_syntax -- Check the PHP syntax of (and execute) the specified file

Description

bool **php_check_syntax** (string `$filename` [, string `&$error_message`])

Performs a syntax (lint) check on the specified *`filename`* testing for scripting errors.

This is similar to using *php -l* from the commandline except that this function will execute (but not output) the checked *`filename`*.

For example, if a function is defined in *`filename`*, this defined function will be available to the file that executed php_check_syntax(), but output from *`filename`* will be suppressed.

Note
For technical reasons, this function is deprecated and removed from PHP. Instead, use *php -l somefile.php* from the commandline.

Parameters

`filename`
: The name of the file being checked.

`error_message`
: If the *`error_message`* parameter is used, it will contain the error message generated by the syntax check. *`error_message`* is passed by reference.

Return Values

Returns **TRUE** if the lint check passed, and **FALSE** if the link check failed or if *`filename`* cannot be opened.

ChangeLog

Version	Description
5.0.5	This function was removed from PHP.
5.0.3	Calling exit() after php_check_syntax()

	resulted in a Segfault.
5.0.1	*error_message* is passed by reference.

Examples

```
php -l somefile.php
```

The above example will output something similar to:

```
PHP Parse error: unexpected T_STRING in /tmp/somefile.php on line 81
```

See Also

- **include()**
- is_readable()

php_strip_whitespace

php_strip_whitespace -- Return source with stripped comments and whitespace

Description

string **php_strip_whitespace** (string `$filename`)

Returns the PHP source code in *`filename`* with PHP comments and whitespace removed. This may be useful for determining the amount of actual code in your scripts compared with the amount of comments. This is similar to using *php -w* from the commandline.

Parameters

`filename`
Path to the PHP file.

Return Values

The stripped source code will be returned on success, or an empty string on failure.

Note

This function works as described as of PHP 5.0.1. Before this it would only return an empty string. For more information on this bug and its prior behavior, see bug report » #29606.

Examples

Example #11 - php_strip_whitespace() example

```
<?php
// PHP comment here

/*
* Another PHP comment
*/

echo        php_strip_whitespace(__FILE__);
// Newlines are considered whitespace, and are removed too:
do_nothing();
?>
```

The above example will output:

```
<?php
echo php_strip_whitespace(__FILE__); do_nothing(); ?>
```

Notice the PHP comments are gone, as are the whitespace and newline after the first echo statement.

show_source

show_source -- Alias of highlight_file()

Description

This function is an alias of: highlight_file().

sleep

sleep -- Delay execution

Description

int **sleep** (int `$seconds`)

Delays the program execution for the given number of *`seconds`*.

Parameters

`seconds`
: Halt time in seconds.

Return Values

Returns zero on success, or **FALSE** on errors.

Errors/Exceptions

If the specified number of *`seconds`* is negative, this function will generate a **E_WARNING**.

Examples

Example #12 - sleep() example

```
<?php

// current time
echo date('h:i:s') . "\n";

// sleep for 10 seconds
sleep(10);

// wake up !
echo date('h:i:s') . "\n";

?>
```

This example will output (after 10 seconds)

```
05:31:23
05:31:33
```

See Also

usleep(), set_time_limit()

sys_getloadavg

sys_getloadavg -- Gets system load average

Description

array **sys_getloadavg** (void)

Returns three samples representing the average system load (the number of processes in the system run queue) over the last 1, 5 and 15 minutes, respectively.

Return Values

Returns an array with three samples (last 1, 5 and 15 minutes).

Examples

Example #13 - A sys_getloadavg() example

```
<?php
$load = sys_getloadavg();
if ($load[0] > 80) {
   header('HTTP/1.1 503 Too busy, try again later');
   die('Server too busy. Please try again later.');
}
?>
```

Notes

Note

This function is not implemented on Windows platforms.

time_nanosleep

time_nanosleep -- Delay for a number of seconds and nanoseconds

Description

mixed **time_nanosleep** (int `$seconds`, int `$nanoseconds`)

Delays program execution for the given number of *seconds* and *nanoseconds*.

Parameters

seconds
: Must be a positive integer.

nanoseconds
: Must be a positive integer less than 1 billion.

Return Values

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

If the delay was interrupted by a signal, an associative array will be returned with the components:

- *seconds* - number of seconds remaining in the delay
- *nanoseconds* - number of nanoseconds remaining in the delay

Examples

Example #14 - time_nanosleep() example

```
<?php
// Careful! This won't work as expected if an array is returned
if (time_nanosleep(0, 500000000)) {
   echo "Slept for half a second.\n";
}

// This is better:
if (time_nanosleep(0, 500000000) === true) {
   echo "Slept for half a second.\n";
}

// And this is the best:
```

```
$nano = time_nanosleep(2, 100000);

if ($nano === true) {
   echo "Slept for 2 seconds, 100 milliseconds.\n";
} elseif ($nano === false) {
   echo "Sleeping failed.\n";
} elseif (is_array($nano)) {
   $seconds = $nano['seconds'];
   $nanoseconds = $nano['nanoseconds'];
   echo "Interrupted by a signal.\n";
   echo "Time remaining: $seconds seconds, $nanoseconds nanoseconds.";
}
?>
```

Notes

Note
This function is not implemented on Windows platforms.

See Also

sleep(), usleep(), set_time_limit()

time_sleep_until

time_sleep_until -- Make the script sleep until the specified time

Description

bool **time_sleep_until** (float `$timestamp`)

Makes the script sleep until the specified *`timestamp`*.

Parameters

`timestamp`
The timestamp when the script should wake.

Return Values

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

Errors/Exceptions

If the specified *`timestamp`* is in the past, this function will generate a **E_WARNING**.

Examples

Example #15 - A time_sleep_until() example

```
<?php

//returns false and generates a warning
var_dump(time_sleep_until(time()-1));

// may only work on faster computers, will sleep up to 0.2 seconds
var_dump(time_sleep_until(time()+0.2));

?>
```

Notes

Note

All signals will be delivered after the script wakes up.

Note
This function is not implemented on Windows platforms.

See Also

- sleep()
- usleep()
- time_nanosleep()

uniqid

uniqid -- Generate a unique ID

Description

string **uniqid** ([string `$prefix` [, bool `$more_entropy`]])

Gets a prefixed unique identifier based on the current time in microseconds.

Parameters

`prefix`
: Can be useful, for instance, if you generate identifiers simultaneously on several hosts that might happen to generate the identifier at the same microsecond. With an empty *`prefix`*, the returned string will be 13 characters long. If *`more_entropy`* is **TRUE**, it will be 23 characters.

`more_entropy`
: If set to **TRUE**, uniqid() will add additional entropy (using the combined linear congruential generator) at the end of the return value, which should make the results more unique.

Return Values

Returns the unique identifier, as a string.

Examples

If you need a unique identifier or token and you intend to give out that token to the user via the network (i.e. session cookies), it is recommended that you use something along these lines:

This will create a 32 character identifier (a 128 bit hex number) that is extremely difficult to predict.

Example #16 - uniqid() Example

```
<?php
// no prefix
// works only in PHP 5 and later versions
$token = md5(uniqid());

// better, difficult to guess
$better_token = md5(uniqid(rand(), true));
?>
```

ChangeLog

Version	Description
5.0.0	The *prefix* parameter was made optional.
4.3.1	The limit of 114 characters long for *prefix* was raised.

unpack

unpack -- Unpack data from binary string

Description

array **unpack** (string `$format`, string `$data`)

Unpacks from a binary string into an array according to the given *`format`*.

unpack() works slightly different from Perl as the unpacked data is stored in an associative array. To accomplish this you have to name the different format codes and separate them by a slash /.

Parameters

`format`
: See pack() for an explanation of the format codes.

`data`
: The packed data.

Return Values

Returns an associative array containing unpacked elements of binary string.

Examples

Example #17 - unpack() example

```
<?php
$array = unpack("c2chars/nint", $binarydata);
?>
```

The resulting array will contain the entries "chars1", "chars2" and "int".

Notes

Caution

Note that PHP internally stores integral values as signed. If you unpack a large unsigned long and it is of the same size as PHP internally stored values the result will

be a negative number even though unsigned unpacking was specified.

See Also

- pack()

usleep

usleep -- Delay execution in microseconds

Description

void **usleep** (int `$micro_seconds`)

Delays program execution for the given number of micro seconds.

Parameters

`micro_seconds`
: Halt time in micro seconds. A micro second is one millionth of a second.

Return Values

No value is returned.

ChangeLog

Version	Description
5.0.0	This function now works on Windows systems.

Examples

Example #18 - usleep() example

```
<?php

// Current time
echo date('h:i:s') . "\n";

// wait for 2 seconds
usleep(2000000);

// back!
echo date('h:i:s') . "\n";

?>
```

The above example will output:

```
11:13:28
11:13:30
```

See Also

- sleep()
- set_time_limit()