Function Handling

Introduction

These functions all handle various operations involved in working with functions.

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

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Function handling Functions

call_user_func_array

call_user_func_array -- Call a user function given with an array of parameters

Description

mixed **call_user_func_array** (callback `$function`, array `$param_arr`)

Call a user defined *`function`* with the parameters in *`param_arr`*.

Parameters

`function`
The function to be called.

`param_arr`
The parameters to be passed to the function, as an indexed array.

Return Values

Returns the function result, or **FALSE** on error.

Examples

Example #1 - call_user_func_array() example

```
<?php
function debug($var, $val)
{
   echo "***DEBUGGING\nVARIABLE: $var\nVALUE:";
   if (is_array($val) || is_object($val) || is_resource($val)) {
       print_r($val);
   } else {
       echo "\n$val\n";
   }
   echo "***\n";
}

$c = mysql_connect();
$host = $_SERVER["SERVER_NAME"];

call_user_func_array('debug', array("host", $host));
call_user_func_array('debug', array("c", $c));
call_user_func_array('debug', array("_POST", $_POST));
?>
```

Example #2 - call_user_func_array() using namespace name

```
<?php

namespace Foobar;

class Foo {
   static public function test($name) {
       print "Hello {$name}!\n";
   }
}

// As of PHP 5.3.0
call_user_func_array(__NAMESPACE__ .'::Foo::test', array('Hannes'));
// Hello Hannes!

// As of PHP 5.3.0
call_user_func_array(array(__NAMESPACE__ .'::Foo', 'test'),
array('Philip'));
// Hello Philip!

?>
```

Notes

Note

Referenced variables in *param_arr* are passed to the function by a reference, others are passed by a value. In other words, it does not depend on the function signature whether the parameter is passed by a value or by a reference.

See Also

- call_user_func()
- information about the callback type

call_user_func

call_user_func -- Call a user function given by the first parameter

Description

mixed **call_user_func** (callback `$function` [, mixed `$parameter` [, mixed `$...`]])

Call a user defined function given by the *`function`* parameter.

Parameters

`function`
The function to be called. Class methods may also be invoked statically using this function by passing *array($classname, $methodname)* to this parameter.

`parameter`
Zero or more parameters to be passed to the function.

Note

Note that the parameters for call_user_func() are not passed by reference.

```
<?php
function increment(&$var)
{
   $var++;
}

$a = 0;
call_user_func('increment', $a);
echo $a; // 0

call_user_func_array('increment', array(&$a)); // You can use this
instead
echo $a; // 1
?>
```

Return Values

Returns the function result, or **FALSE** on error.

Examples

Example #3 - call_user_func() example

```
<?php
function barber($type)
{
   echo "You wanted a $type haircut, no problem";
}
call_user_func('barber', "mushroom");
call_user_func('barber', "shave");
?>
```

Example #4 - call_user_func() using namespace name

```
<?php

namespace Foobar;

class Foo {
   static public function test() {
       print "Hello world!\n";
   }
}

call_user_func(__NAMESPACE__ .'::Foo::test'); // As of PHP 5.3.0
// Hello world!
call_user_func(array(__NAMESPACE__ .'::Foo', 'test')); // As of PHP 5.3.0
// Hello world!

?>
```

Example #5 - Using a class method

```
<?php

class myclass {
   static function say_hello()
   {
       echo "Hello!\n";
   }
}

$classname = "myclass";

call_user_func(array($classname, 'say_hello'));
call_user_func($classname .'::say_hello'); // As of 5.2.3

?>
```

See Also

- call_user_func_array()
- is_callable()
- information about the callback type

create_function

create_function -- Create an anonymous (lambda-style) function

Description

string **create_function** (string `$args`, string `$code`)

Creates an anonymous function from the parameters passed, and returns a unique name for it.

Parameters

Usually these parameters will be passed as single quote delimited strings. The reason for using single quoted strings, is to protect the variable names from parsing, otherwise, if you use double quotes there will be a need to escape the variable names, e.g. *\$avar*.

`args`
: The function arguments.

`code`
: The function code.

Return Values

Returns a unique function name as a string, or **FALSE** on error.

Examples

Example #6 - Creating an anonymous function with create_function()

You can use this function, to (for example) create a function from information gathered at run time:

```
<?php
$newfunc = create_function('$a,$b', 'return "ln($a) + ln($b) = " . log($a *
$b);');
echo "New anonymous function: $newfunc\n";
echo $newfunc(2, M_E) . "\n";
// outputs
// New anonymous function: lambda_1
// ln(2) + ln(2.718281828459) = 1.6931471805599
?>
```

Or, perhaps to have general handler function that can apply a set of operations to a list of parameters:

Example #7 - Making a general processing function with create_function()

```
<?php
function process($var1, $var2, $farr)
{
   foreach ($farr as $f) {
       echo $f($var1, $var2) . "\n";
   }
}

// create a bunch of math functions
$f1 = 'if ($a >=0) {return "b*a^2 = ".$b*sqrt($a);} else {return false;}';
$f2 = "return \"min(b^2+a, a^2,b) = \".min(\$a*\$a+\$b,\$b*\$b+\$a);";
$f3 = 'if ($a > 0 && $b != 0) {return "ln(a)/b = ".log($a)/$b; } else {
return false; }';
$farr = array(
   create_function('$x,$y', 'return "some trig: ".(sin($x) + $x*cos($y));'),
   create_function('$x,$y', 'return "a hypotenuse: ".sqrt($x*$x + $y*$y);'),
   create_function('$a,$b', $f1),
   create_function('$a,$b', $f2),
   create_function('$a,$b', $f3)
   );

echo "\nUsing the first array of anonymous functions\n";
echo "parameters: 2.3445, M_PI\n";
process(2.3445, M_PI, $farr);

// now make a bunch of string processing functions
$garr = array(
   create_function('$b,$a', 'if (strncmp($a, $b, 3) == 0) return "** \"$a\"
'.
   'and \"$b\"\n** Look the same to me! (looking at the first 3 chars)";'),
   create_function('$a,$b', '; return "CRCs: " . crc32($a) . " ,
".crc32(b);'),
   create_function('$a,$b', '; return "similar(a,b) = " . similar_text($a,
$b, &$p) . "($p%)";')
   );
echo "\nUsing the second array of anonymous functions\n";
process("Twas brilling and the slithy toves", "Twas the night", $garr);
?>
```

The above example will output:

```
Using the first array of anonymous functions
parameters: 2.3445, M_PI
some trig: -1.6291725057799
a hypotenuse: 3.9199852871011
b*a^2 = 4.8103313314525
min(b^2+a, a^2,b) = 8.6382729035898
ln(a/b) = 0.27122299212594

Using the second array of anonymous functions
** "Twas the night" and "Twas brilling and the slithy toves"
** Look the same to me! (looking at the first 3 chars)
CRCs: -725381282 , 1908338681
similar(a,b) = 11(45.833333333333%)
```

But perhaps the most common use for of lambda-style (anonymous) functions is to create callback functions, for example when using array_walk() or usort()

Example #8 - Using anonymous functions as callback functions

```
<?php
$av = array("the ", "a ", "that ", "this ");
array_walk($av, create_function('&$v,$k', '$v = $v . "mango";'));
print_r($av);
?>
```

The above example will output:

```
Array
(
 [0] => the mango
 [1] => a mango
 [2] => that mango
 [3] => this mango
)
```

an array of strings ordered from shorter to longer

```
<?php

$sv = array("small", "larger", "a big string", "it is a string thing");
print_r($sv);

?>
```

The above example will output:

```
Array
(
 [0] => small
 [1] => larger
 [2] => a big string
 [3] => it is a string thing
)
```

sort it from longer to shorter

```
<?php

usort($sv, create_function('$a,$b','return strlen($b) - strlen($a);'));
print_r($sv);

?>
```

The above example will output:

```
Array
(
 [0] => it is a string thing
 [1] => a big string
 [2] => larger
```

```
 [3] => small
)
```

func_get_arg

func_get_arg -- Return an item from the argument list

Description

mixed **func_get_arg** (int `$arg_num`)

Gets the specified argument from a user-defined function's argument list.

This function may be used in conjunction with func_get_args() and func_num_args() to allow user-defined functions to accept variable-length argument lists.

Parameters

`arg_num`
The argument offset. Function arguments are counted starting from zero.

Return Values

Returns the specified argument, or **FALSE** on error.

ChangeLog

Version	Description
5.3.0	This function can now be used in parameter lists.

Errors/Exceptions

Generates a warning if called from outside of a user-defined function, or if *`arg_num`* is greater than the number of arguments actually passed.

Examples

Example #9 - func_get_arg() example

```
<?php
function foo()
```

```
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br />\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg(1) . "<br />\n";
    }
}

foo (1, 2, 3);
?>
```

Notes

Note
Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If this value must be passed, the results should be assigned to a variable, and that variable should be passed.

Note
This function returns a copy of the passed arguments only, and does not account for default (non-passed) arguments.

See Also

- func_get_args()
- func_num_args()

func_get_args

func_get_args -- Returns an array comprising a function's argument list

Description

array **func_get_args** (void)

Gets an array of the function's argument list.

This function may be used in conjunction with func_get_arg() and func_num_args() to allow user-defined functions to accept variable-length argument lists.

Return Values

Returns an array in which each element is a copy of the corresponding member of the current user-defined function's argument list.

ChangeLog

Version	Description
5.3.0	This function can now be used in parameter lists.

Errors/Exceptions

Generates a warning if called from outside of a user-defined function.

Examples

Example #10 - func_get_args() example

```
<?php
function foo()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br />\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg(1) . "<br />\n";
    }
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo "Argument $i is: " . $arg_list[$i] . "<br />\n";
```

```
    }
}

foo(1, 2, 3);
?>
```

Notes

Note
Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If this value must be passed, the results should be assigned to a variable, and that variable should be passed.

Note
This function returns a copy of the passed arguments only, and does not account for default (non-passed) arguments.

See Also

- func_get_arg()
- func_num_args()

func_num_args

func_num_args -- Returns the number of arguments passed to the function

Description

int **func_num_args** (void)

Gets the number of arguments passed to the function.

This function may be used in conjunction with func_get_arg() and func_get_args() to allow user-defined functions to accept variable-length argument lists.

Return Values

Returns the number of arguments passed into the current user-defined function.

ChangeLog

Version	Description
5.3.0	This function can now be used in parameter lists.

Errors/Exceptions

Generates a warning if called from outside of a user-defined function.

Examples

Example #11 - func_num_args() example

```
<?php
function foo()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs\n";
}

foo(1, 2, 3);    // Prints 'Number of arguments: 3'
?>
```

Notes

Note
Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If this value must be passed, the results should be assigned to a variable, and that variable should be passed.

See Also

- func_get_arg()
- func_get_args()

function_exists

function_exists -- Return **TRUE** if the given function has been defined

Description

bool **function_exists** (string `$function_name`)

Checks the list of defined functions, both built-in (internal) and user-defined, for *`function_name`*.

Parameters

`function_name`
: The function name, as a string.

Return Values

Returns **TRUE** if *`function_name`* exists and is a function, **FALSE** otherwise.

Note
This function will return **FALSE** for constructs, such as **include_once()** and echo().

Examples

Example #12 - function_exists() example

```
<?php
if (function_exists('imap_open')) {
   echo "IMAP functions are available.<br />\n";
} else {
   echo "IMAP functions are not available.<br />\n";
}
?>
```

Notes

Note
A function name may exist even if the function itself is unusable due to configuration or compiling options (with the image functions being an example).

See Also

- method_exists()
- is_callable()
- get_defined_functions()

get_defined_functions

get_defined_functions -- Returns an array of all defined functions

Description

array **get_defined_functions** (void)

Gets an array of all defined functions.

Return Values

Returns an multidimensional array containing a list of all defined functions, both built-in (internal) and user-defined. The internal functions will be accessible via *$arr["internal"]*, and the user defined ones using *$arr["user"]* (see example below).

Examples

Example #13 - get_defined_functions() **example**

```
<?php
function myrow($id, $data)
{
   return "<tr><th>$id</th><td>$data</td></tr>\n";
}

$arr = get_defined_functions();

print_r($arr);
?>
```

The above example will output something similar to:

```
Array
(
   [internal] => Array
       (
           [0] => zend_version
           [1] => func_num_args
           [2] => func_get_arg
           [3] => func_get_args
           [4] => strlen
           [5] => strcmp
           [6] => strncmp
           ...
           [750] => bcscale
           [751] => bccomp
       )

   [user] => Array
       (
```

```
            [0] => myrow
        )

)
```

See Also

- function_exists()
- get_defined_vars()
- get_defined_constants()
- get_declared_classes()

register_shutdown_function

register_shutdown_function -- Register a function for execution on shutdown

Description

void **register_shutdown_function** (callback `$function` [, mixed `$parameter` [, mixed `$...`]])

Registers the function named by *`function`* to be executed when script processing is complete.

Multiple calls to register_shutdown_function() can be made, and each will be called in the same order as they were registered. If you call exit() within one registered shutdown function, processing will stop completely and no other registered shutdown functions will be called.

In PHP 4.0.6 and earlier under Apache, the registered shutdown functions are called after the request has been completed (including sending any output buffers), so it is not possible to send output to the browser using echo() or print(), or retrieve the contents of any output buffers using ob_get_contents(). Since PHP 4.1, the shutdown functions are called as the part of the request so that it's possible to send the output from them. There is currently no way to process the data with output buffering functions in the shutdown function. Shutdown function is called after closing all opened output buffers thus, for example, its output will not be compressed if zlib.output_compression is enabled.

As of PHP 4, it is possible to pass parameters to the shutdown function by passing additional parameters to register_shutdown_function().

Parameters

`function`

`parameter`

`...`

Return Values

No value is returned.

Notes

Note
Typically undefined functions cause fatal errors in PHP, but when the *`function`* called with register_shutdown_function() is undefined, an error of level **E_WARNING** is generated instead. Also, for reasons internal to PHP, this error will refer to *Unknown* at line #0.

Note
Working directory of the script can change inside the shutdown function under some web servers, e.g. Apache.

Note
Shutdown function is called during the script shutdown so headers are always already sent.

See Also

- auto_append_file
- exit()
- The section on connection handling

register_tick_function

register_tick_function -- Register a function for execution on each tick

Description

bool **register_tick_function** (callback `$function` [, mixed `$arg` [, mixed `$...`]])

Registers the given *`function`* to be executed when a tick is called.

Parameters

`function`
The function name as a string, or an array consisting of an object and a method.

`arg`

`...`

Return Values

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

Examples

Example #14 - register_tick_function() example

```
<?php
// using a function as the callback
register_tick_function('my_function', true);

// using an object->method
$object = new my_class();
register_tick_function(array(&$object, 'my_method'), true);
?>
```

Notes

Warning
register_tick_function() should not be used with threaded web server modules. Ticks are not working in ZTS mode and may crash your web server.

See Also

- declare
- unregister_tick_function()

unregister_tick_function

unregister_tick_function -- De-register a function for execution on each tick

Description

void **unregister_tick_function** (string `$function_name`)

De-registers the function named by *`function_name`* so it is no longer executed when a tick is called.

Parameters

`function_name`
: The function name, as a string.

Return Values

No value is returned.

See Also

- register_tick_function()