

Function Handling Functions

Introduzione

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

Requisiti

Non sono necessarie librerie esterne per utilizzare questo modulo.

Installazione

Non è necessaria nessuna installazione per usare queste funzioni, esse fanno parte del core di PHP.

Configurazione di Runtime

Questa estensione non definisce alcuna direttiva di configurazione in `php.ini`

Costanti predefinite

Questa estensione non definisce alcuna costante.

Sommario

[call_user_func_array](#) -- Call a user function given with an array of parameters

[call_user_func](#) -- Call a user function given by the first parameter

[create_function](#) -- Create an anonymous (lambda-style) function

[func_get_arg](#) -- Return an item from the argument list

[func_get_args](#) -- Returns an array comprising a function's argument list

[func_num_args](#) -- Returns the number of arguments passed to the function

[function_exists](#) -- Return TRUE if the given function has been defined

[get_defined_functions](#) -- Returns an array of all defined functions

[register_shutdown_function](#) -- Register a function for execution on shutdown

[register_tick_function](#) -- Register a function for execution on each tick

[unregister_tick_function](#) -- De-register a function for execution on each tick

call_user_func_array

(PHP 4 >= 4.0.4, PHP 5)

`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 given by *function*, with the parameters in *param_arr*. For example:

Esempio 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));
?>
```

See also [call_user_func\(\)](#), and information about the [callback](#) type

call_user_func

(PHP 3 >= 3.0.3, PHP 4, PHP 5)

`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. Take the following:

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

Object methods may also be invoked statically using this function by passing *array(\$ObjectName, \$methodName)* to the *function* parameter.

```
<?php
class myclass {
    function say_hello()
    {
        echo "Hello!\n";
    }
}

$classname = "myclass";

call_user_func(array($classname, 'say_hello'));
?>
```

Nota: 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
?>
```

See also: [is_callable\(\)](#), [call_user_func_array\(\)](#), e information about the [callback](#) type.

create_function

(PHP 4 >= 4.0.1, PHP 5)

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. Usually the *args* will be passed as a single quote delimited string, and this is also recommended for the *code*. 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`.

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

Esempio 1. Creating an anonymous function with `create_function()`

```
<?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:

Esempio 2. 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) . " (
);
echo "\nUsing the second array of anonymous functions\n";
process("Twas brillling and the slithy toves", "Twas the night", $garr);
?>
```

and when you run the code above, the output will be:

```
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\(\)](#)

Esempio 3. 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);
?>
```

outputs:

```
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);
?>
```

outputs:

```
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);

?>
```

outputs:

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

func_get_arg

(PHP 4, PHP 5)

func_get_arg -- Return an item from the argument list

Description

mixed **func_get_arg** (int *arg_num*)

Returns the argument which is at the *arg_num*'th offset into a user-defined function's argument list. Function arguments are counted starting from zero. **func_get_arg()** will generate a warning if called from outside of a function definition. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.

If *arg_num* is greater than the number of arguments actually passed, a warning will be generated and **func_get_arg()** will return **FALSE**.

Nota: Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If you must pass this value, assign the results to a variable, and pass the variable.

```
<?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);
?>
```

func_get_arg() may be used in conjunction with [func_num_args\(\)](#) and [func_get_args\(\)](#) to allow user-defined functions to accept variable-length argument lists.

func_get_args

(PHP 4, PHP 5)

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

Description

array **func_get_args** (void)

Returns an array in which each element is a copy of the corresponding member of the current user-defined function's argument list. **func_get_args()** will generate a warning if called from outside of a function definition. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.

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

Nota: Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If you must pass this value, assign the results to a variable, and pass the variable.

```
<?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);
?>
```

func_get_args() may be used in conjunction with [func_num_args\(\)](#) and [func_get_arg\(\)](#) to allow user-defined functions to accept variable-length argument lists.

func_num_args

(PHP 4, PHP 5)

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

Description

int **func_num_args** (void)

Returns the number of arguments passed into the current user-defined function. **func_num_args()** will generate a warning if called from outside of a user-defined function. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.

Nota: Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If you must pass this value, assign the results to a variable, and pass the variable.

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

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

func_num_args() may be used in conjunction with [func_get_arg\(\)](#) and [func_get_args\(\)](#) to allow user-defined functions to accept variable-length argument lists.

function_exists

(PHP 3 >= 3.0.7, PHP 4, PHP 5)

`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*. Restituisce TRUE in caso di successo, FALSE in caso di fallimento.

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

Note that 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). Also note that `function_exists()` will return `FALSE` for constructs, such as [include_once\(\)](#) and [echo\(\)](#).

See also [method_exists\(\)](#), [is_callable\(\)](#) and [get_defined_functions\(\)](#).

get_defined_functions

(PHP 4 >= 4.0.4, PHP 5)

`get_defined_functions` -- Returns an array of all defined functions

Description

array `get_defined_functions` (void)

This function 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).

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

$arr = get_defined_functions();

print_r($arr);
?>
```

Will output something along the lines of:

```
Array
(
    [internal] => Array
        (
            [0] => zend_version
            [1] => func_num_args
            [2] => func_get_arg
            [3] => func_get_args
            [4] => strlen
            [5] => strcmp
```

```
[6] => strcmp
...
[750] => bcscale
[751] => bccomp
)

[user] => Array
(
    [0] => myrow
)

)
```

See also [function_exists\(\)](#), [get_defined_vars\(\)](#) and [get_defined_constants\(\)](#).

register_shutdown_function

(PHP 3 >= 3.0.4, PHP 4, PHP 5)

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()**.

Nota: 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.

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

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

See also [auto_append_file](#), [exit\(\)](#), and the section on [connection handling](#).

register_tick_function

(PHP 4 >= 4.0.3, PHP 5)

register_tick_function -- Register a function for execution on each tick

Description

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

Registers the function named by *func* to be executed when a [tick](#) is called. Also, you may pass an array consisting of an object and a method as the *func*.

Esempio 1. 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);
?>
```

Avvertimento

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

See also [declare](#) and [unregister_tick_function\(\)](#).

unregister_tick_function

(PHP 4 >= 4.0.3, PHP 5)

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.