Strings

Introduction

These functions all manipulate strings in various ways. Some more specialized sections can be found in the regular expression and URL handling sections.

For information on how strings behave, especially with regard to usage of single quotes, double quotes, and escape sequences, see the Strings entry in the Types section of the manual.

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

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.

CRYPT_SALT_LENGTH integer

CRYPT_STD_DES integer

CRYPT_EXT_DES integer

CRYPT_MD5 integer

CRYPT_BLOWFISH integer

HTML_SPECIALCHARS (integer)

HTML_ENTITIES (integer)

ENT_COMPAT (integer)

ENT_QUOTES (integer)

ENT_NOQUOTES (integer)

CHAR_MAX (integer)

LC_CTYPE (integer)

LC_NUMERIC (integer)

LC_TIME (integer)

LC_COLLATE (integer)

LC_MONETARY (integer)

LC_ALL (integer)

LC_MESSAGES (integer)

STR_PAD_LEFT (integer)

STR_PAD_RIGHT (integer)

STR_PAD_BOTH (integer)

String Functions

See Also

For even more powerful string handling and manipulating functions take a look at the POSIX regular expression functions and the Perl compatible regular expression functions.

addcslashes

addcslashes -- Quote string with slashes in a C style

Description

string **addcslashes** (string `$str`, string `$charlist`)

Returns a string with backslashes before characters that are listed in *`charlist`* parameter.

Parameters

`str`
: The string to be escaped.

`charlist`
: A list of characters to be escaped. If *`charlist`* contains characters *n*, *r* etc., they are converted in C-like style, while other non-alphanumeric characters with ASCII codes lower than 32 and higher than 126 converted to octal representation. When you define a sequence of characters in the charlist argument make sure that you know what characters come between the characters that you set as the start and end of the range.

```
<?php
echo addcslashes('foo[ ]', 'A..z');
// output:  \f\o\o\[ \]
// All upper and lower-case letters will be escaped
// ... but so will the [\]^_` and any tabs, line
// feeds, carriage returns, etc.
?>
```

Also, if the first character in a range has a higher ASCII value than the second character in the range, no range will be constructed. Only the start, end and period characters will be escaped. Use the ord() function to find the ASCII value for a character.

```
<?php
echo addcslashes("zoo['.']", 'z..A');
// output:  \zoo['\.']
?>
```

Be careful if you choose to escape characters 0, a, b, f, n, r, t and v. They will be converted to \0, \a, \b, \f, \n, \r, \t and \v. In PHP \0 (NULL), \r (carriage return), \n (newline), \f (form feed), \v (vertical tab) and \t (tab) are predefined escape sequences, while in C all of these are predefined escape sequences.

Return Values

Returns the escaped string.

ChangeLog

Version	Description
5.2.5	The escape sequences \v and \f were added.

Examples

charlist like "\0..\37", which would escape all characters with ASCII code between 0 and 31.

Example #1 - addcslashes() example

```
<?php
$escaped = addcslashes($not_escaped, "\0..\37!@\177..\377");
?>
```

See Also

- stripcslashes()
- stripslashes()
- addslashes()
- htmlspecialchars()
- quotemeta()

addslashes

addslashes -- Quote string with slashes

Description

string **addslashes** (string `$str`)

Returns a string with backslashes before characters that need to be quoted in database queries etc. These characters are single quote ('), double quote ("), backslash (\ ) and NUL (the **NULL** byte).

An example use of addslashes() is when you're entering data into a database. For example, to insert the name *O'reilly* into a database, you will need to escape it. Most databases do this with a \ which would mean *O\'reilly*. This would only be to get the data into the database, the extra \ will not be inserted. Having the PHP directive magic_quotes_sybase set to *on* will mean ' is instead escaped with another '.

The PHP directive magic_quotes_gpc is *on* by default, and it essentially runs addslashes() on all GET, POST, and COOKIE data. Do not use addslashes() on strings that have already been escaped with magic_quotes_gpc as you'll then do double escaping. The function get_magic_quotes_gpc() may come in handy for checking this.

Parameters

`str`
The string to be escaped.

Return Values

Returns the escaped string.

Examples

Example #2 - An addslashes() example

```
<?php
$str = "Is your name O'reilly?";

// Outputs: Is your name O\'reilly?
echo addslashes($str);
?>
```

See Also

- stripcslashes()
- stripslashes()
- addcslashes()
- htmlspecialchars()
- quotemeta()
- get_magic_quotes_gpc()

bin2hex

bin2hex -- Convert binary data into hexadecimal representation

Description

string **bin2hex** (string `$str`)

Returns an ASCII string containing the hexadecimal representation of *`str`*. The conversion is done byte-wise with the high-nibble first.

Parameters

`str`
: A character.

Return Values

Returns the hexadecimal representation of the given string.

See Also

- pack()
- unpack()

chop

chop -- Alias of rtrim()

Description

This function is an alias of: rtrim().

Notes

Note
chop() is different than the Perl *chop()* function, which removes the last character in the string.

chr

chr -- Return a specific character

Description

string **chr** (int `$ascii`)

Returns a one-character string containing the character specified by *`ascii`*.

This function complements ord().

Parameters

`ascii`
: The ascii code.

Return Values

Returns the specified character.

Examples

Example #3 - chr() example

```
<?php
$str = "The string ends in escape: ";
$str .= chr(27); /* add an escape character at the end of $str */

/* Often this is more useful */

$str = sprintf("The string ends in escape: %c", 27);
?>
```

See Also

- sprintf() with a format string of *%c*
- ord()
- An » ASCII-table

chunk_split

chunk_split -- Split a string into smaller chunks

Description

string **chunk_split** (string `$body` [, int `$chunklen` [, string `$end`]])

Can be used to split a string into smaller chunks which is useful for e.g. converting base64_encode() output to match RFC 2045 semantics. It inserts *`end`* every *`chunklen`* characters.

Parameters

`body`
: The string to be chunked.

`chunklen`
: The chunk length. Defaults to 76.

`end`
: The line ending sequence. Defaults to "\r\n".

Return Values

Returns the chunked string.

Examples

Example #4 - chunk_split() example

```
<?php
// format $data using RFC 2045 semantics
$new_string = chunk_split(base64_encode($data));
?>
```

See Also

- str_split()
- explode()
- split()

- wordwrap()
- » RFC 2045

convert_cyr_string

convert_cyr_string -- Convert from one Cyrillic character set to another

Description

string **convert_cyr_string** (string `$str`, string `$from`, string `$to`)

Converts from one Cyrillic character set to another.

Parameters

`str`
: The string to be converted.

`from`
: The source Cyrillic character set, as a single character.

`to`
: The target Cyrillic character set, as a single character.

Supported characters are:

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

Return Values

Returns the converted string.

Notes

Note
This function is binary-safe.

convert_uudecode

convert_uudecode -- Decode a uuencoded string

Description

string **convert_uudecode** (string `$data`)

convert_uudecode() decodes a uuencoded string.

Parameters

`data`
: The uuencoded data.

Return Values

Returns the decoded data as a string.

Examples

Example #5 - convert_uudecode() example

```
<?php
/* Can you imagine what this will print? :) */
echo convert_uudecode("+22!L;W9E(%!(4\"$`\n`");
?>
```

See Also

- convert_uuencode()

convert_uuencode

convert_uuencode -- Uuencode a string

Description

string **convert_uuencode** (string `$data`)

convert_uuencode() encodes a string using the uuencode algorithm.

Uuencode translates all strings (including binary's ones) into printable characters, making them safe for network transmissions. Uuencoded data is about 35% larger than the original.

Parameters

`data`
The data to be encoded.

Return Values

Returns the uuencoded data.

Examples

Example #6 - convert_uuencode() example

```
<?php
$some_string = "test\ntext text\r\n";

echo convert_uuencode($some_string);
?>
```

See Also

- convert_uudecode()
- base64_encode()

count_chars

count_chars -- Return information about characters used in a string

Description

mixed **count_chars** (string `$string` [, int `$mode`])

Counts the number of occurrences of every byte-value (0..255) in *`string`* and returns it in various ways.

Parameters

`string`
The examined string.

`mode`
The optional parameter *`mode`* defaults to 0.

Return Values

Depending on *`mode`* count_chars() returns one of the following:

- 0 - an array with the byte-value as key and the frequency of every byte as value.
- 1 - same as 0 but only byte-values with a frequency greater than zero are listed.
- 2 - same as 0 but only byte-values with a frequency equal to zero are listed.
- 3 - a string containing all unique characters is returned.
- 4 - a string containing all not used characters is returned.

Examples

Example #7 - count_chars() example

```
<?php
$data = "Two Ts and one F.";

foreach (count_chars($data, 1) as $i => $val) {
  echo "There were $val instance(s) of \"" , chr($i) , "\" in the
string.\n";
}
?>
```

The above example will output:

```
There were 4 instance(s) of " " in the string.
There were 1 instance(s) of "." in the string.
There were 1 instance(s) of "F" in the string.
There were 2 instance(s) of "T" in the string.
There were 1 instance(s) of "a" in the string.
There were 1 instance(s) of "d" in the string.
There were 1 instance(s) of "e" in the string.
There were 2 instance(s) of "n" in the string.
There were 2 instance(s) of "o" in the string.
There were 1 instance(s) of "s" in the string.
There were 1 instance(s) of "w" in the string.
```

See Also

- strpos()
- substr_count()

crc32

crc32 -- Calculates the crc32 polynomial of a string

Description

int **crc32** (string `$str`)

Generates the cyclic redundancy checksum polynomial of 32-bit lengths of the *`str`*. This is usually used to validate the integrity of data being transmitted.

Because PHP's integer type is signed, and many crc32 checksums will result in negative integers, you need to use the "%u" formatter of sprintf() or printf() to get the string representation of the unsigned crc32 checksum.

Parameters

`str`
The data.

Return Values

Returns the crc32 checksum of *`str`* as an integer.

Examples

Example #8 - Displaying a crc32 checksum

This example shows how to print a converted checksum with the printf() function:

```
<?php
$checksum = crc32("The quick brown fox jumped over the lazy dog.");
printf("%u\n", $checksum);
?>
```

See Also

- md5()
- sha1()

crypt

crypt -- One-way string encryption (hashing)

Description

string **crypt** (string `$str` [, string `$salt`])

crypt() will return an encrypted string using the standard Unix DES -based encryption algorithm or alternative algorithms that may be available on the system.

Some operating systems support more than one type of encryption. In fact, sometimes the standard DES-based encryption is replaced by an MD5-based encryption algorithm. The encryption type is triggered by the salt argument. At install time, PHP determines the capabilities of the crypt function and will accept salts for other encryption types. If no salt is provided, PHP will auto-generate a standard two character salt by default, unless the default encryption type on the system is MD5, in which case a random MD5-compatible salt is generated. PHP sets a constant named **CRYPT_SALT_LENGTH** which tells you whether a regular two character salt applies to your system or the longer twelve character salt is applicable.

The standard DES-based encryption crypt() returns the salt as the first two characters of the output. It also only uses the first eight characters of *`str`*, so longer strings that start with the same eight characters will generate the same result (when the same salt is used).

On systems where the crypt() function supports multiple encryption types, the following constants are set to 0 or 1 depending on whether the given type is available:

- **CRYPT_STD_DES** - Standard DES-based encryption with a two character salt
- **CRYPT_EXT_DES** - Extended DES-based encryption with a nine character salt
- **CRYPT_MD5** - MD5 encryption with a twelve character salt starting with 1
- **CRYPT_BLOWFISH** - Blowfish encryption with a sixteen character salt starting with 2 or $2a$

Parameters

`str`
: The string to be encrypted.

`salt`
: An optional salt string to base the encryption on. If not provided, one will be randomly generated by PHP each time you call this function. If you are using the supplied salt, you should be aware that the salt is generated once. If you are calling this function repeatedly, this may impact both appearance and security.

Return Values

Returns the encrypted string.

Examples

Example #9 - crypt() examples

```
<?php
$password = crypt('mypassword'); // let the salt be automatically generated

/* You should pass the entire results of crypt() as the salt for comparing a
  password, to avoid problems when different hashing algorithms are used.
(As
  it says above, standard DES-based password hashing uses a 2-character
salt,
  but MD5-based hashing uses 12.) */
if (crypt($user_input, $password) == $password) {
  echo "Password verified!";
}
?>
```

Example #10 - Using crypt() with htpasswd

```
<?php
// Set the password
$password = 'mypassword';

// Get the hash, letting the salt be automatically generated
$hash = crypt($password);
?>
```

Example #11 - Using crypt() with different encryption types

```
<?php
if (CRYPT_STD_DES == 1) {
   echo 'Standard DES: ' . crypt('rasmuslerdorf', 'rl') . "\n";
}

if (CRYPT_EXT_DES == 1) {
   echo 'Extended DES: ' . crypt('rasmuslerdorf', '_J9..rasm') . "\n";
}

if (CRYPT_MD5 == 1) {
   echo 'MD5:          ' . crypt('rasmuslerdorf', '$1$rasmusle$') . "\n";
}

if (CRYPT_BLOWFISH == 1) {
   echo 'Blowfish:     ' . crypt('rasmuslerdorf',
'$2a$07$rasmuslerd..........$') . "\n";
}
```

```
?>
```

The above example will output something similar to:

```
Standard DES: rl.3StKT.4T8M
Extended DES: _J9..rasmBYk8r9AiWNc
MD5:          $1$rasmusle$rISCgZzpwk3UhDidwXvin0
Blowfish:     $2a$07$rasmuslerd............nIdrcHdxcUxWomQX9j6kvERCFjTg7Ra
```

Notes

Note
There is no decrypt function, since crypt() uses a one-way algorithm.

See Also

- md5()
- The Mcrypt extension
- The Unix man page for your crypt function for more information

echo

echo -- Output one or more strings

Description

void **echo** (string `$arg1` [, string `$...`])

Outputs all parameters.

echo() is not actually a function (it is a language construct), so you are not required to use parentheses with it. echo() (unlike some other language constructs) does not behave like a function, so it cannot always be used in the context of a function. Additionally, if you want to pass more than one parameter to echo(), the parameters must not be enclosed within parentheses.

echo() also has a shortcut syntax, where you can immediately follow the opening tag with an equals sign. This short syntax only works with the short_open_tag configuration setting enabled.

```
I have <?=$foo?> foo.
```

Parameters

arg1
The parameter to output.

...

Return Values

No value is returned.

Examples

Example #12 - echo() examples

```
<?php
echo "Hello World";

echo "This spans
multiple lines. The newlines will be
output as well";
```

```
echo "This spans\nmultiple lines. The newlines will be\noutput as well.";

echo "Escaping characters is done \"Like this\".";

// You can use variables inside of an echo statement
$foo = "foobar";
$bar = "barbaz";

echo "foo is $foo"; // foo is foobar

// You can also use arrays
$baz = array("value" => "foo");

echo "this is {$baz['value']} !"; // this is foo !

// Using single quotes will print the variable name, not the value
echo 'foo is $foo'; // foo is $foo

// If you are not using any other characters, you can just echo variables
echo $foo;          // foobar
echo $foo,$bar;     // foobarbarbaz

// Some people prefer passing multiple parameters to echo over
concatenation.
echo 'This ', 'string ', 'was ', 'made ', 'with multiple parameters.',
chr(10);
echo 'This ' . 'string ' . 'was ' . 'made ' . 'with concatenation.' . "\n";

echo <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
END;

// Because echo does not behave like a function, the following code is
invalid.
($some_var) ? echo 'true' : echo 'false';

// However, the following examples will work:
($some_var) ? print 'true' : print 'false'; // print is also a construct,
but
                                           // it behaves like a function, so
                                           // it may be used in this
context.
echo $some_var ? 'true': 'false'; // changing the statement around
?>
```

Notes

For a short discussion about the differences between print() and echo(), see this FAQTs Knowledge Base Article: » http://www.faqts.com/knowledge_base/view.phtml/aid/1/fid/40

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

See Also

- print()
- printf()
- flush()

explode

explode -- Split a string by string

Description

array **explode** (string `$delimiter`, string `$string` [, int `$limit`])

Returns an array of strings, each of which is a substring of *`string`* formed by splitting it on boundaries formed by the string *`delimiter`*.

Parameters

`delimiter`
: The boundary string.

`string`
: The input string.

`limit`
: If *`limit`* is set, the returned array will contain a maximum of *`limit`* elements with the last element containing the rest of *`string`*. If the *`limit`* parameter is negative, all components except the last - *`limit`* are returned.

Although implode() can, for historical reasons, accept its parameters in either order, explode() cannot. You must ensure that the *`delimiter`* argument comes before the *`string`* argument.

Return Values

If *`delimiter`* is an empty string (""), explode() will return **FALSE**. If *`delimiter`* contains a value that is not contained in *`string`*, then explode() will return an array containing *`string`*.

ChangeLog

Version	Description
5.1.0	Support for negative *`limit`* s was added
4.0.1	The *`limit`* parameter was added

Examples

Example #13 - explode() examples

```
<?php
// Example 1
$pizza  = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2

// Example 2
$data = "foo:*:1023:1000::/home/foo:/bin/sh";
list($user, $pass, $uid, $gid, $gecos, $home, $shell) = explode(":", $data);
echo $user; // foo
echo $pass; // *

?>
```

Example #14 - *limit* parameter examples

```
<?php
$str = 'one|two|three|four';

// positive limit
print_r(explode('|', $str, 2));

// negative limit (since PHP 5.1)
print_r(explode('|', $str, -1));
?>
```

The above example will output:

```
Array
(
   [0] => one
   [1] => two|three|four
)
Array
(
   [0] => one
   [1] => two
   [2] => three
)
```

Notes

Note
This function is binary-safe.

See Also

- preg_split()
- str_split()
- str_word_count()
- strtok()
- implode()

fprintf

fprintf -- Write a formatted string to a stream

Description

int **fprintf** (resource `$handle`, string `$format` [, mixed `$args` [, mixed `$...`]])

Write a string produced according to *`format`* to the stream resource specified by *`handle`*.

Parameters

`handle`
A file system pointer resource that is typically created using fopen().

`format`
See sprintf() for a description of *`format`*.

`args`

`...`

Return Values

Returns the length of the string written.

Examples

Example #15 - fprintf(): zero-padded integers

```
<?php
if (!($fp = fopen('date.txt', 'w'))) {
   return;
}

fprintf($fp, "%04d-%02d-%02d", $year, $month, $day);
// will write the formatted ISO date to date.txt
?>
```

Example #16 - fprintf(): formatting currency

```
<?php
```

```
if (!($fp = fopen('currency.txt', 'w'))) {
   return;
}

$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money will output "123.1";
$len = fprintf($fp, '%01.2f', $money);
// will write "123.10" to currency.txt

echo "wrote $len bytes to currency.txt";
// use the return value of fprintf to determine how many bytes we wrote
?>
```

See Also

- printf()
- sprintf()
- sscanf()
- fscanf()
- vsprintf()
- number_format()

get_html_translation_table

get_html_translation_table -- Returns the translation table used by htmlspecialchars() and htmlentities()

Description

array **get_html_translation_table** ([int `$table` [, int `$quote_style`]])

get_html_translation_table() will return the translation table that is used internally for htmlspecialchars() and htmlentities().

Note
Special characters can be encoded in several ways. E.g. " can be encoded as *"*, *"* or *&#x22*. get_html_translation_table() returns only the most common form for them.

Parameters

`table`
: There are two new constants (**HTML_ENTITIES**, **HTML_SPECIALCHARS**) that allow you to specify the table you want. Default value for *`table`* is **HTML_SPECIALCHARS**.

`quote_style`
: Like the htmlspecialchars() and htmlentities() functions you can optionally specify the *`quote_style`* you are working with. The default is **ENT_COMPAT** mode. See the description of these modes in htmlspecialchars().

Return Values

Returns the translation table as an array.

Examples

Example #17 - Translation Table Example

```
<?php
$trans = get_html_translation_table(HTML_ENTITIES);
$str = "Hallo & <Frau> & Krämer";
$encoded = strtr($str, $trans);
?>
```

The *$encoded* variable will now contain: *"Hallo & <Frau> & Krämer".*

See Also

- htmlspecialchars()
- htmlentities()
- html_entity_decode()

hebrev

hebrev -- Convert logical Hebrew text to visual text

Description

string **hebrev** (string `$hebrew_text` [, int `$max_chars_per_line`])

Converts logical Hebrew text to visual text.

The function tries to avoid breaking words.

Parameters

`hebrew_text`
: A Hebrew input string.

`max_chars_per_line`
: This optional parameter indicates maximum number of characters per line that will be returned.

Return Values

Returns the visual string.

See Also

- hebrevc()

hebrevc

hebrevc -- Convert logical Hebrew text to visual text with newline conversion

Description

string **hebrevc** (string `$hebrew_text` [, int `$max_chars_per_line`])

This function is similar to hebrev() with the difference that it converts newlines (\n) to "
\n".

The function tries to avoid breaking words.

Parameters

`hebrew_text`
: A Hebrew input string.

`max_chars_per_line`
: This optional parameter indicates maximum number of characters per line that will be returned.

Return Values

Returns the visual string.

See Also

- hebrev()

html_entity_decode

html_entity_decode -- Convert all HTML entities to their applicable characters

Description

string **html_entity_decode** (string `$string` [, int `$quote_style` [, string `$charset`]])

html_entity_decode() is the opposite of htmlentities() in that it converts all HTML entities to their applicable characters from *`string`*.

Parameters

`string`
The input string.

`quote_style`
The optional second *`quote_style`* parameter lets you define what will be done with 'single' and "double" quotes. It takes on one of three constants with the default being **ENT_COMPAT**:

Available *`quote_style`* constants

Constant Name	Description
ENT_COMPAT	Will convert double-quotes and leave single-quotes alone.
ENT_QUOTES	Will convert both double and single quotes.
ENT_NOQUOTES	Will leave both double and single quotes unconverted.

`charset`
The ISO-8859-1 character set is used as default for the optional third *`charset`*. This defines the character set used in conversion. Following character sets are supported in PHP 4.3.0 and later.

Supported charsets

Charset	Aliases	Description
ISO-8859-1	ISO8859-1	Western European, Latin-1
ISO-8859-15	ISO8859-15	Western European, Latin-9. Adds the Euro sign, French

		and Finnish letters missing in Latin-1(ISO-8859-1).
UTF-8		ASCII compatible multi-byte 8-bit Unicode.
cp866	ibm866, 866	DOS-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1251	Windows-1251, win-1251, 1251	Windows-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1252	Windows-1252, 1252	Windows specific charset for Western European.
KOI8-R	koi8-ru, koi8r	Russian. This charset is supported in 4.3.2.
BIG5	950	Traditional Chinese, mainly used in Taiwan.
GB2312	936	Simplified Chinese, national standard character set.
BIG5-HKSCS		Big5 with Hong Kong extensions, Traditional Chinese.
Shift_JIS	SJIS, 932	Japanese
EUC-JP	EUCJP	Japanese

Note
Any other character sets are not recognized and ISO-8859-1 will be used instead.

Return Values

Returns the decoded string.

ChangeLog

Version	Description
5.0.0	Support for multi-byte character sets was added.

Examples

Example #18 - Decoding HTML entities

```
<?php
$orig = "I'll \"walk\" the <b>dog</b> now";

$a = htmlentities($orig);

$b = html_entity_decode($a);

echo $a; // I'll &quot;walk&quot; the &lt;b&gt;dog&lt;/b&gt; now

echo $b; // I'll "walk" the <b>dog</b> now


// For users prior to PHP 4.3.0 you may do this:
function unhtmlentities($string)
{
   // replace numeric entities
   $string = preg_replace('~&#x([0-9a-f]+);~ei', 'chr(hexdec("\\1"))',
$string);
   $string = preg_replace('~&#([0-9]+);~e', 'chr("\\1")', $string);
   // replace literal entities
   $trans_tbl = get_html_translation_table(HTML_ENTITIES);
   $trans_tbl = array_flip($trans_tbl);
   return strtr($string, $trans_tbl);
}

$c = unhtmlentities($a);

echo $c; // I'll "walk" the <b>dog</b> now

?>
```

Notes

Note

You might wonder why trim(html_entity_decode(' ')); doesn't reduce the string to an empty string, that's because the ' ' entity is not ASCII code 32 (which is stripped by trim()) but ASCII code 160 (0xa0) in the default ISO 8859-1 characterset.

See Also

- htmlentities()
- htmlspecialchars()
- get_html_translation_table()
- urldecode()

htmlentities

htmlentities -- Convert all applicable characters to HTML entities

Description

string **htmlentities** (string `$string` [, int `$quote_style` [, string `$charset` [, bool `$double_encode`]]])

This function is identical to htmlspecialchars() in all ways, except with htmlentities(), all characters which have HTML character entity equivalents are translated into these entities.

If you're wanting to decode instead (the reverse) you can use html_entity_decode().

Parameters

`string`
The input string.

`quote_style`
Like htmlspecialchars(), the optional second *`quote_style`* parameter lets you define what will be done with 'single' and "double" quotes. It takes on one of three constants with the default being **ENT_COMPAT**:

Available *`quote_style`* constants

Constant Name	Description
ENT_COMPAT	Will convert double-quotes and leave single-quotes alone.
ENT_QUOTES	Will convert both double and single quotes.
ENT_NOQUOTES	Will leave both double and single quotes unconverted.

`charset`
Like htmlspecialchars(), it takes an optional third argument *`charset`* which defines character set used in conversion. Presently, the ISO-8859-1 character set is used as the default. Following character sets are supported in PHP 4.3.0 and later.

Supported charsets

Charset	Aliases	Description
ISO-8859-1	ISO8859-1	Western European, Latin-1

ISO-8859-15	ISO8859-15	Western European, Latin-9. Adds the Euro sign, French and Finnish letters missing in Latin-1(ISO-8859-1).
UTF-8		ASCII compatible multi-byte 8-bit Unicode.
cp866	ibm866, 866	DOS-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1251	Windows-1251, win-1251, 1251	Windows-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1252	Windows-1252, 1252	Windows specific charset for Western European.
KOI8-R	koi8-ru, koi8r	Russian. This charset is supported in 4.3.2.
BIG5	950	Traditional Chinese, mainly used in Taiwan.
GB2312	936	Simplified Chinese, national standard character set.
BIG5-HKSCS		Big5 with Hong Kong extensions, Traditional Chinese.
Shift_JIS	SJIS, 932	Japanese
EUC-JP	EUCJP	Japanese

Note
Any other character sets are not recognized and ISO-8859-1 will be used instead.

double_encode

When *double_encode* is turned off PHP will not encode existing html entities. The default is to convert everything.

Return Values

Returns the encoded string.

ChangeLog

Version	Description
5.2.3	The *double_encode* parameter was added.
4.1.0	The *charset* parameter was added.
4.0.3	The *quote_style* parameter was added.

Examples

Example #19 - A htmlentities() example

```
<?php
$str = "A 'quote' is <b>bold</b>";

// Outputs: A 'quote' is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($str);

// Outputs: A &#039;quote&#039; is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($str, ENT_QUOTES);
?>
```

See Also

- html_entity_decode()
- get_html_translation_table()
- htmlspecialchars()
- nl2br()
- urlencode()

htmlspecialchars_decode

htmlspecialchars_decode -- Convert special HTML entities back to characters

Description

string **htmlspecialchars_decode** (string `$string` [, int `$quote_style`])

This function is the opposite of htmlspecialchars(). It converts special HTML entities back to characters.

The converted entities are: *&*, *"* (when ENT_NOQUOTES is not set), *'* (when ENT_QUOTES is set), *<* and *>*.

Parameters

`string`
The string to decode

`quote_style`
The quote style. One of the following constants:

quote_style constants

Constant Name	Description
ENT_COMPAT	Will convert double-quotes and leave single-quotes alone (default)
ENT_QUOTES	Will convert both double and single quotes
ENT_NOQUOTES	Will leave both double and single quotes unconverted

Return Values

Returns the decoded string.

Examples

Example #20 - A htmlspecialchars_decode() example

```
<?php
$str = '<p>this -&gt; &quot;</p>';

echo htmlspecialchars_decode($str);

// note that here the quotes aren't converted
echo htmlspecialchars_decode($str, ENT_NOQUOTES);
?>
```

The above example will output:

```
<p>this -> "</p>
<p>this -> &quot;</p>
```

See Also

- htmlspecialchars()
- html_entity_decode()
- get_html_translation_table()

htmlspecialchars

htmlspecialchars -- Convert special characters to HTML entities

Description

string **htmlspecialchars** (string `$string` [, int `$quote_style` [, string `$charset` [, bool `$double_encode`]]])

Certain characters have special significance in HTML, and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with some of these conversions made; the translations made are those most useful for everyday web programming. If you require all HTML character entities to be translated, use htmlentities() instead.

This function is useful in preventing user-supplied text from containing HTML markup, such as in a message board or guest book application.

The translations performed are:

- '&' (ampersand) becomes '&'
- '"' (double quote) becomes '"' when **ENT_NOQUOTES** is not set.
- ''' (single quote) becomes ''' only when **ENT_QUOTES** is set.
- '<' (less than) becomes '<'
- '>' (greater than) becomes '>'

Parameters

`string`
The string being converted.

`quote_style`
The optional second argument, *`quote_style`*, tells the function what to do with single and double quote characters. The default mode, **ENT_COMPAT**, is the backwards compatible mode which only translates the double-quote character and leaves the single-quote untranslated. If **ENT_QUOTES** is set, both single and double quotes are translated and if **ENT_NOQUOTES** is set neither single nor double quotes are translated.

`charset`
Defines character set used in conversion. The default character set is ISO-8859-1. Following character sets are supported in PHP 4.3.0 and later.

Supported charsets

Charset	Aliases	Description

ISO-8859-1	ISO8859-1	Western European, Latin-1
ISO-8859-15	ISO8859-15	Western European, Latin-9. Adds the Euro sign, French and Finnish letters missing in Latin-1(ISO-8859-1).
UTF-8		ASCII compatible multi-byte 8-bit Unicode.
cp866	ibm866, 866	DOS-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1251	Windows-1251, win-1251, 1251	Windows-specific Cyrillic charset. This charset is supported in 4.3.2.
cp1252	Windows-1252, 1252	Windows specific charset for Western European.
KOI8-R	koi8-ru, koi8r	Russian. This charset is supported in 4.3.2.
BIG5	950	Traditional Chinese, mainly used in Taiwan.
GB2312	936	Simplified Chinese, national standard character set.
BIG5-HKSCS		Big5 with Hong Kong extensions, Traditional Chinese.
Shift_JIS	SJIS, 932	Japanese
EUC-JP	EUCJP	Japanese

Note
Any other character sets are not recognized and ISO-8859-1 will be used instead.

`double_encode`

When *`double_encode`* is turned off PHP will not encode existing html entities, the default is to convert everything.

Return Values

The converted string.

ChangeLog

Version	Description
5.2.3	The *double_encode* parameter was added.
4.1.0	The *charset* parameter was added.

Examples

Example #21 - htmlspecialchars() example

```
<?php
$new = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
echo $new; // &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
?>
```

Notes

Note

Note that this function does not translate anything beyond what is listed above. For full entity translation, see htmlentities().

See Also

- get_html_translation_table()
- htmlspecialchars_decode()
- strip_tags()
- htmlentities()
- nl2br()

implode

implode -- Join array elements with a string

Description

string **implode** (string `$glue`, array `$pieces`)

Join array elements with a *`glue`* string.

Note
implode() can, for historical reasons, accept its parameters in either order. For consistency with explode(), however, it may be less confusing to use the documented order of arguments.

Parameters

`glue`
: Defaults to an empty string. This is not the preferred usage of implode() as *`glue`* would be the second parameter and thus, the bad prototype would be used.

`pieces`
: The array of strings to implode.

Return Values

Returns a string containing a string representation of all the array elements in the same order, with the glue string between each element.

ChangeLog

Version	Description
4.3.0	The *`glue`* parameter became optional.

Examples

Example #22 - implode() example

```
<?php

$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);

echo $comma_separated; // lastname,email,phone

?>
```

Notes

Note

This function is binary-safe.

See Also

- explode()
- split()

join

join -- Alias of implode()

Description

This function is an alias of: implode().

lcfirst

lcfirst -- Make a string's first character lowercase

Description

string **lcfirst** (string `$str`)

Returns a string with the first character of *`str`*, lowercased if that character is alphabetic.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

Parameters

`str`
The input string.

Return Values

Returns the resulting string.

Examples

Example #23 - lcfirst() example

```
<?php
$foo = 'HelloWorld';
$foo = lcfirst($foo);             // helloWorld

$bar = 'HELLO WORLD!';
$bar = lcfirst($bar);             // hELLO WORLD!
$bar = lcfirst(strtoupper($bar)); // hELLO WORLD!
?>
```

See Also

- ucfirst()
- strtolower()
- strtoupper()
- ucwords()

levenshtein

levenshtein -- Calculate Levenshtein distance between two strings

Description

int **levenshtein** (string `$str1`, string `$str2`)

int **levenshtein** (string `$str1`, string `$str2`, int `$cost_ins`, int `$cost_rep`, int `$cost_del`)

The Levenshtein distance is defined as the minimal number of characters you have to replace, insert or delete to transform *`str1`* into *`str2`*. The complexity of the algorithm is *O(m*n)*, where *n* and *m* are the length of *`str1`* and *`str2`* (rather good when compared to similar_text(), which is O(max(n,m)**3), but still expensive).

In its simplest form the function will take only the two strings as parameter and will calculate just the number of insert, replace and delete operations needed to transform *`str1`* into *`str2`*.

A second variant will take three additional parameters that define the cost of insert, replace and delete operations. This is more general and adaptive than variant one, but not as efficient.

Parameters

`str1`
One of the strings being evaluated for Levenshtein distance.

`str2`
One of the strings being evaluated for Levenshtein distance.

`cost_ins`
Defines the cost of insertion.

`cost_rep`
Defines the cost of replacement.

`cost_del`
Defines the cost of deletion.

Return Values

This function returns the Levenshtein-Distance between the two argument strings or -1, if one of the argument strings is longer than the limit of 255 characters.

Examples

Example #24 - levenshtein() example

```
<?php
// input misspelled word
$input = 'carrrot';

// array of words to check against
$words  = array('apple','pineapple','banana','orange',
                'radish','carrot','pea','bean','potato');

// no shortest distance found, yet
$shortest = -1;

// loop through words to find the closest
foreach ($words as $word) {

   // calculate the distance between the input word,
   // and the current word
   $lev = levenshtein($input, $word);

   // check for an exact match
   if ($lev == 0) {

       // closest word is this one (exact match)
       $closest = $word;
       $shortest = 0;

       // break out of the loop; we've found an exact match
       break;
   }

   // if this distance is less than the next found shortest
   // distance, OR if a next shortest word has not yet been found
   if ($lev <= $shortest || $shortest < 0) {
       // set the closest match, and shortest distance
       $closest  = $word;
       $shortest = $lev;
   }
}

echo "Input word: $input\n";
if ($shortest == 0) {
   echo "Exact match found: $closest\n";
} else {
   echo "Did you mean: $closest?\n";
}

?>
```

The above example will output:

```
Input word: carrrot
Did you mean: carrot?
```

See Also

- soundex()
- similar_text()
- metaphone()

localeconv

localeconv -- Get numeric formatting information

Description

array **localeconv** (void)

Returns an associative array containing localized numeric and monetary formatting information.

Return Values

localeconv() returns data based upon the current locale as set by setlocale(). The associative array that is returned contains the following fields:

Array element	**Description**
decimal_point	Decimal point character
thousands_sep	Thousands separator
grouping	Array containing numeric groupings
int_curr_symbol	International currency symbol (i.e. USD)
currency_symbol	Local currency symbol (i.e. $)
mon_decimal_point	Monetary decimal point character
mon_thousands_sep	Monetary thousands separator
mon_grouping	Array containing monetary groupings
positive_sign	Sign for positive values
negative_sign	Sign for negative values
int_frac_digits	International fractional digits
frac_digits	Local fractional digits
p_cs_precedes	**TRUE** if currency_symbol precedes a positive value, **FALSE** if it succeeds one
p_sep_by_space	**TRUE** if a space separates currency_symbol from a positive value, **FALSE** otherwise

n_cs_precedes	**TRUE** if currency_symbol precedes a negative value, **FALSE** if it succeeds one
n_sep_by_space	**TRUE** if a space separates currency_symbol from a negative value, **FALSE** otherwise
p_sign_posn	• 0 - Parentheses surround the quantity and currency_symbol • 1 - The sign string precedes the quantity and currency_symbol • 2 - The sign string succeeds the quantity and currency_symbol • 3 - The sign string immediately precedes the currency_symbol • 4 - The sign string immediately succeeds the currency_symbol
n_sign_posn	• 0 - Parentheses surround the quantity and currency_symbol • 1 - The sign string precedes the quantity and currency_symbol • 2 - The sign string succeeds the quantity and currency_symbol • 3 - The sign string immediately precedes the currency_symbol • 4 - The sign string immediately succeeds the currency_symbol

The *p_sign_posn*, and *n_sign_posn* contain a string of formatting options. Each number representing one of the above listed conditions.

The grouping fields contain arrays that define the way numbers should be grouped. For example, the monetary grouping field for the nl_NL locale (in UTF-8 mode with the euro sign), would contain a 2 item array with the values 3 and 3. The higher the index in the array, the farther left the grouping is. If an array element is equal to **CHAR_MAX**, no further grouping is done. If an array element is equal to 0, the previous element should be used.

Examples

Example #25 - localeconv() example

```
<?php
if (false !== setlocale(LC_ALL, 'nl_NL.UTF-8@euro')) {
   $locale_info = localeconv();
   print_r($locale_info);
}
?>
```

The above example will output:

```
Array
(
   [decimal_point] => .
   [thousands_sep] =>
   [int_curr_symbol] => EUR
   [currency_symbol] => ?
   [mon_decimal_point] => ,
   [mon_thousands_sep] =>
   [positive_sign] =>
   [negative_sign] => -
   [int_frac_digits] => 2
   [frac_digits] => 2
   [p_cs_precedes] => 1
   [p_sep_by_space] => 1
   [n_cs_precedes] => 1
   [n_sep_by_space] => 1
   [p_sign_posn] => 1
   [n_sign_posn] => 2
   [grouping] => Array
       (
       )

   [mon_grouping] => Array
       (
           [0] => 3
           [1] => 3
       )

)
```

See Also

- setlocale()

ltrim

ltrim -- Strip whitespace (or other characters) from the beginning of a string

Description

string **ltrim** (string `$str` [, string `$charlist`])

Strip whitespace (or other characters) from the beginning of a string.

Parameters

`str`
: The input string.

`charlist`
: You can also specify the characters you want to strip, by means of the *`charlist`* parameter. Simply list all characters that you want to be stripped. With.. you can specify a range of characters.

Return Values

This function returns a string with whitespace stripped from the beginning of *`str`*. Without the second parameter, ltrim() will strip these characters:

- " " (ASCII *32* (*0x20*)), an ordinary space.
- "\t" (ASCII *9* (*0x09*)), a tab.
- "\n" (ASCII *10* (*0x0A*)), a new line (line feed).
- "\r" (ASCII *13* (*0x0D*)), a carriage return.
- "\0" (ASCII *0* (*0x00*)), the *NUL* -byte.
- "\x0B" (ASCII *11* (*0x0B*)), a vertical tab.

ChangeLog

Version	Description
4.1.0	The *`charlist`* parameter was added.

Examples

Example #26 - Usage example of ltrim()

```
<?php

$text = "\t\tThese are a few words :) ...  ";
$binary = "\x09Example string\x0A";
$hello  = "Hello World";
var_dump($text, $binary, $hello);

print "\n";


$trimmed = ltrim($text);
var_dump($trimmed);

$trimmed = ltrim($text, " \t.");
var_dump($trimmed);

$trimmed = ltrim($hello, "Hdle");
var_dump($trimmed);

// trim the ASCII control characters at the beginning of $binary
// (from 0 to 31 inclusive)
$clean = ltrim($binary, "\x00..\x1F");
var_dump($clean);

?>
```

The above example will output:

```
string(32) "        These are a few words :) ...  "
string(16) "    Example string
"
string(11) "Hello World"

string(30) "These are a few words :) ...  "
string(30) "These are a few words :) ...  "
string(7) "o World"
string(15) "Example string
"
```

See Also

- trim()
- rtrim()

md5_file

md5_file -- Calculates the md5 hash of a given file

Description

string **md5_file** (string `$filename` [, bool `$raw_output`])

Calculates the MD5 hash of the file specified by the *`filename`* parameter using the » RSA Data Security, Inc. MD5 Message-Digest Algorithm, and returns that hash. The hash is a 32-character hexadecimal number.

Parameters

`filename`
: The filename

`raw_output`
: When **TRUE**, returns the digest in raw binary format with a length of 16. Defaults to **FALSE**.

Return Values

Returns a string on success, **FALSE** otherwise.

ChangeLog

Version	Description
5.0.0	Added the *`raw_output`* parameter
5.1.0	Changed the function to use the streams API. It means that you can use it with wrappers, like *md5_file('http://example.com/..')*

See Also

- md5()
- sha1_file()

- crc32()

md5

md5 -- Calculate the md5 hash of a string

Description

string **md5** (string `$str` [, bool `$raw_output`])

Calculates the MD5 hash of *`str`* using the » RSA Data Security, Inc. MD5 Message-Digest Algorithm, and returns that hash.

Parameters

`str`
: The string.

`raw_output`
: If the optional *`raw_output`* is set to **TRUE**, then the md5 digest is instead returned in raw binary format with a length of 16. Defaults to **FALSE**.

Return Values

Returns the hash as a 32-character hexadecimal number.

ChangeLog

Version	Description
5.0.0	The *`raw_output`* parameter was added.

Examples

Example #27 - A md5() example

```
<?php
$str = 'apple';

if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f') {
    echo "Would you like a green or red apple?";
    exit;
}
```

```
?>
```

See Also

- sha1_file()
- crc32()
- sha1()

metaphone

metaphone -- Calculate the metaphone key of a string

Description

string **metaphone** (string `$str` [, int `$phones`])

Calculates the metaphone key of *`str`*.

Similar to soundex() metaphone creates the same key for similar sounding words. It's more accurate than soundex() as it knows the basic rules of English pronunciation. The metaphone generated keys are of variable length.

Metaphone was developed by Lawrence Philips <lphilips at verity dot com>. It is described in ["Practical Algorithms for Programmers", Binstock & Rex, Addison Wesley, 1995].

Parameters

`str`
: The input string.

`phones`

Return Values

Returns the metaphone key as a string.

money_format

money_format -- Formats a number as a currency string

Description

string **money_format** (string `$format`, float `$number`)

money_format() returns a formatted version of *`number`*. This function wraps the C library function **strfmon()**, with the difference that this implementation converts only one number at a time.

Parameters

`format`
The format specification consists of the following sequence:

- a *%* character
- optional flags
- optional field width
- optional left precision
- optional right precision
- a required conversion character

Flags

One or more of the optional flags below can be used:
= *f*
The character = followed by a (single byte) character *f* to be used as the numeric fill character. The default fill character is space.

^
Disable the use of grouping characters (as defined by the current locale).

+ or *(*
Specify the formatting style for positive and negative numbers. If + is used, the locale's equivalent for + and - will be used. If *(* is used, negative amounts are enclosed in parenthesis. If no specification is given, the default is +.

!
Suppress the currency symbol from the output string.

-
If present, it will make all fields left-justified (padded to the right), as opposed to the

default which is for the fields to be right-justified (padded to the left).

Field width

w
A decimal digit string specifying a minimum field width. Field will be right-justified unless the flag - is used. Default value is 0 (zero).

Left precision

 # n
The maximum number of digits (*n*) expected to the left of the decimal character (e.g. the decimal point). It is used usually to keep formatted output aligned in the same columns, using the fill character if the number of digits is less than *n*. If the number of actual digits is bigger than *n*, then this specification is ignored. If grouping has not been suppressed using the ^ flag, grouping separators will be inserted before the fill characters (if any) are added. Grouping separators will not be applied to fill characters, even if the fill character is a digit. To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign symbols are padded as necessary with space characters to make their positive and negative formats an equal length.

Right precision

. p
A period followed by the number of digits (*p*) after the decimal character. If the value of *p* is 0 (zero), the decimal character and the digits to its right will be omitted. If no right precision is included, the default will dictated by the current local in use. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion characters

i
The number is formatted according to the locale's international currency format (e.g. for the USA locale: USD 1,234.56).

n
: The number is formatted according to the locale's national currency format (e.g. for the de_DE locale: DM1.234,56).

%
: Returns the % character.

number
: The number to be formatted.

Return Values

Returns the formatted string. Characters before and after the formatting string will be returned unchanged.

Notes

Note
The function money_format() is only defined if the system has strfmon capabilities. For example, Windows does not, so money_format() is undefined in Windows.

Note
The **LC_MONETARY** category of the locale settings, affects the behavior of this function. Use setlocale() to set to the appropriate default locale before using this function.

Examples

Example #28 - money_format() Example

We will use different locales and format specifications to illustrate the use of this function.

```
<?php

$number = 1234.56;

// let's print the international format for the en_US locale
setlocale(LC_MONETARY, 'en_US');
echo money_format('%i', $number) . "\n";
```

```
// USD 1,234.56

// Italian national format with 2 decimals`
setlocale(LC_MONETARY, 'it_IT');
echo money_format('%.2n', $number) . "\n";
// L. 1.234,56

// Using a negative number
$number = -1234.5672;

// US national format, using () for negative numbers
// and 10 digits for left precision
setlocale(LC_MONETARY, 'en_US');
echo money_format('%(#10n', $number) . "\n";
// ($        1,234.57)

// Similar format as above, adding the use of 2 digits of right
// precision and '*' as a fill character
echo money_format('%=*(#10.2n', $number) . "\n";
// ($********1,234.57)

// Let's justify to the left, with 14 positions of width, 8 digits of
// left precision, 2 of right precision, withouth grouping character
// and using the international format for the de_DE locale.
setlocale(LC_MONETARY, 'de_DE');
echo money_format('%=*^-14#8.2i', 1234.56) . "\n";
// DEM 1234,56****

// Let's add some blurb before and after the conversion specification
setlocale(LC_MONETARY, 'en_GB');
$fmt = 'The final value is %i (after a 10%% discount)';
echo money_format($fmt, 1234.56) . "\n";
// The final value is  GBP 1,234.56 (after a 10% discount)

?>
```

See Also

- setlocale()
- sscanf()
- sprintf()
- printf()
- number_format()

nl_langinfo

nl_langinfo -- Query language and locale information

Description

string **nl_langinfo** (int `$item`)

nl_langinfo() is used to access individual elements of the locale categories. Unlike localeconv(), which returns all of the elements, nl_langinfo() allows you to select any specific element.

Parameters

`item`

`item` may be an integer value of the element or the constant name of the element. The following is a list of constant names for *`item`* that may be used and their description. Some of these constants may not be defined or hold no value for certain locales.

nl_langinfo Constants

Constant	Description
LC_TIME Category Constants	
ABDAY_(1-7)	Abbreviated name of n-th day of the week.
DAY_(1-7)	Name of the n-th day of the week (DAY_1 = Sunday).
ABMON_(1-12)	Abbreviated name of the n-th month of the year.
MON_(1-12)	Name of the n-th month of the year.
AM_STR	String for Ante meridian.
PM_STR	String for Post meridian.
D_T_FMT	String that can be used as the format string for strftime() to represent time and date.
D_FMT	String that can be used as the format string for strftime() to represent date.
T_FMT	String that can be used as the format string for strftime() to represent time.

T_FMT_AMPM	String that can be used as the format string for strftime() to represent time in 12-hour format with ante/post meridian.
ERA	Alternate era.
ERA_YEAR	Year in alternate era format.
ERA_D_T_FMT	Date and time in alternate era format (string can be used in strftime()).
ERA_D_FMT	Date in alternate era format (string can be used in strftime()).
ERA_T_FMT	Time in alternate era format (string can be used in strftime()).
LC_MONETARY Category Constants	
INT_CURR_SYMBOL	International currency symbol.
CURRENCY_SYMBOL	Local currency symbol.
CRNCYSTR	Same value as CURRENCY_SYMBOL.
MON_DECIMAL_POINT	Decimal point character.
MON_THOUSANDS_SEP	Thousands separator (groups of three digits).
MON_GROUPING	Like 'grouping' element.
POSITIVE_SIGN	Sign for positive values.
NEGATIVE_SIGN	Sign for negative values.
INT_FRAC_DIGITS	International fractional digits.
FRAC_DIGITS	Local fractional digits.
P_CS_PRECEDES	Returns 1 if CURRENCY_SYMBOL precedes a positive value.
P_SEP_BY_SPACE	Returns 1 if a space separates CURRENCY_SYMBOL from a positive value.
N_CS_PRECEDES	Returns 1 if CURRENCY_SYMBOL precedes a negative value.
N_SEP_BY_SPACE	Returns 1 if a space separates CURRENCY_SYMBOL from a negative

	value.
P_SIGN_POSN	• Returns 0 if parentheses surround the quantity and currency_symbol. • Returns 1 if the sign string precedes the quantity and currency_symbol. • Returns 2 if the sign string follows the quantity and currency_symbol. • Returns 3 if the sign string immediately precedes the currency_symbol. • Returns 4 if the sign string immediately follows the currency_symbol.
N_SIGN_POSN	
LC_NUMERIC Category Constants	
DECIMAL_POINT	Decimal point character.
RADIXCHAR	Same value as DECIMAL_POINT.
THOUSANDS_SEP	Separator character for thousands (groups of three digits).
THOUSEP	Same value as THOUSANDS_SEP.
GROUPING	
LC_MESSAGES Category Constants	
YESEXPR	Regex string for matching 'yes' input.
NOEXPR	Regex string for matching 'no' input.
YESSTR	Output string for 'yes'.
NOSTR	Output string for 'no'.
LC_CTYPE Category Constants	
CODESET	Return a string with the name of the character encoding.

Return Values

Returns the element as a string, or **FALSE** if *item* is not valid.

Notes

Note
This function is not implemented on Windows platforms.

See Also

- setlocale()
- localeconv()

nl2br

nl2br -- Inserts HTML line breaks before all newlines in a string

Description

string **nl2br** (string `$string`)

Returns *`string`* with '<br />' inserted before all newlines.

Parameters

`string`
The input string.

Return Values

Returns the altered string.

ChangeLog

Version	Description
4.0.5	nl2br() is now XHTML compliant. All older versions will return *`string`* with ' ' inserted before newlines instead of ' '.

Examples

Example #29 - using nl2br()

```
<?php
echo nl2br("foo isn't\n bar");
?>
```

The above example will output:

```
foo isn't<br />
bar
```

See Also

- htmlspecialchars()
- htmlentities()
- wordwrap()
- str_replace()

number_format

number_format -- Format a number with grouped thousands

Description

string **number_format** (float `$number` [, int `$decimals`])

string **number_format** (float `$number`, int `$decimals`, string `$dec_point`, string `$thousands_sep`)

This function accepts either one, two, or four parameters (not three):

If only one parameter is given, *`number`* will be formatted without decimals, but with a comma (",") between every group of thousands.

If two parameters are given, *`number`* will be formatted with *`decimals`* decimals with a dot (".") in front, and a comma (",") between every group of thousands.

If all four parameters are given, *`number`* will be formatted with *`decimals`* decimals, *`dec_point`* instead of a dot (".") before the decimals and *`thousands_sep`* instead of a comma (",") between every group of thousands.

Return Values

A formatted version of *`number`*.

Parameters

`number`
: The number being formatted.

`decimals`
: Sets the number of decimal points.

`dec_point`
: Sets the separator for the decimal point.

`thousands_sep`
: Sets the thousands separator. Only the first character of *`thousands_sep`* is used. For example, if you use *bar* as *`thousands_sep`* on the number *1000*, number_format() will return *1b000*.

Examples

Example #30 - number_format() Example

For instance, French notation usually use two decimals, comma (',') as decimal separator, and space (' ') as thousand separator. This is achieved with this line :

```
<?php

$number = 1234.56;

// english notation (default)
$english_format_number = number_format($number);
// 1,235

// French notation
$nombre_format_francais = number_format($number, 2, ',', ' ');
// 1 234,56

$number = 1234.5678;

// english notation without thousands seperator
$english_format_number = number_format($number, 2, '.', '');
// 1234.57

?>
```

See Also

- money_format()
- sprintf()
- printf()
- sscanf()

ord

ord -- Return ASCII value of character

Description

int **ord** (string `$string`)

Returns the ASCII value of the first character of *`string`*.

This function complements chr().

Parameters

`string`
A character.

Return Values

Returns the ASCII value as an integer.

Examples

Example #31 - ord() example

```
<?php
$str = "\n";
if (ord($str) == 10) {
   echo "The first character of \$str is a line feed.\n";
}
?>
```

See Also

- chr()
- An » ASCII-table

parse_str

parse_str -- Parses the string into variables

Description

void **parse_str** (string `$str` [, array `&$arr`])

Parses *`str`* as if it were the query string passed via a URL and sets variables in the current scope.

Note
To get the current *QUERY_STRING*, you may use the variable $_SERVER['QUERY_STRING']. Also, you may want to read the section on variables from external sources.

Note
The magic_quotes_gpc setting affects the output of this function, as parse_str() uses the same mechanism that PHP uses to populate the *$_GET*, *$_POST*, etc. variables.

Parameters

`str`
: The input string.

`arr`
: If the second parameter *`arr`* is present, variables are stored in this variable as array elements instead.

Return Values

No value is returned.

ChangeLog

Version	Description
4.0.3	The *`arr`* parameter was added

Examples

Example #32 - Using parse_str()

```
<?php
$str = "first=value&arr[]=foo+bar&arr[]=baz";
parse_str($str);
echo $first;  // value
echo $arr[0]; // foo bar
echo $arr[1]; // baz

parse_str($str, $output);
echo $output['first'];  // value
echo $output['arr'][0]; // foo bar
echo $output['arr'][1]; // baz

?>
```

See Also

- parse_url()
- pathinfo()
- http_build_query()
- get_magic_quotes_gpc()
- urldecode()

print

print -- Output a string

Description

int **print** (string `$arg`)

Outputs *`arg`*.

print() is not actually a real function (it is a language construct) so you are not required to use parentheses with its argument list.

For a short discussion about the differences between print() and echo(), see this FAQTs Knowledge Base Article: » http://www.faqts.com/knowledge_base/view.phtml/aid/1/fid/40

Parameters

`arg`
The input data.

Return Values

Returns *1*, always.

Examples

Example #33 - print() examples

```
<?php
print("Hello World");

print "print() also works without parentheses.";

print "This spans
multiple lines. The newlines will be
output as well";

print "This spans\nmultiple lines. The newlines will be\noutput as well.";

print "escaping characters is done \"Like this\".";

// You can use variables inside of a print statement
$foo = "foobar";
$bar = "barbaz";

print "foo is $foo"; // foo is foobar
```

```
// You can also use arrays
$bar = array("value" => "foo");

print "this is {$bar['value']} !"; // this is foo !

// Using single quotes will print the variable name, not the value
print 'foo is $foo'; // foo is $foo

// If you are not using any other characters, you can just print variables
print $foo;          // foobar

print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
?>
```

Notes

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

See Also

- echo()
- printf()
- flush()

printf

printf -- Output a formatted string

Description

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

Produces output according to *`format`*.

Parameters

`format`
: See sprintf() for a description of *`format`*.

`args`

`...`

Return Values

Returns the length of the outputted string.

See Also

- print()
- sprintf()
- vprintf()
- sscanf()
- fscanf()
- flush()

quoted_printable_decode

quoted_printable_decode -- Convert a quoted-printable string to an 8 bit string

Description

string **quoted_printable_decode** (string `$str`)

This function returns an 8-bit binary string corresponding to the decoded quoted printable string (according to » RFC2045, section 6.7, not » RFC2821, section 4.5.2, so additional periods are not stripped from the beginning of line).

This function is similar to imap_qprint(), except this one does not require the IMAP module to work.

Parameters

`str`
The input string.

Return Values

Returns the 8-bit binary string.

quotemeta

quotemeta -- Quote meta characters

Description

string **quotemeta** (string `$str`)

Returns a version of str with a backslash character (\ ) before every character that is among these:
`. \ + * ? [ ^ ] ( $ )`

Parameters

`str`
The input string.

Return Values

Returns the string with meta characters quoted.

Notes

Note
This function is binary-safe.

See Also

- addslashes()
- addcslashes()
- htmlentities()
- htmlspecialchars()
- nl2br()
- stripslashes()
- stripcslashes()
- ereg()

rtrim

rtrim -- Strip whitespace (or other characters) from the end of a string

Description

string **rtrim** (string `$str` [, string `$charlist`])

This function returns a string with whitespace stripped from the end of *`str`*.

Without the second parameter, rtrim() will strip these characters:

- " " (ASCII *32* (*0x20*)), an ordinary space.
- "\t" (ASCII *9* (*0x09*)), a tab.
- "\n" (ASCII *10* (*0x0A*)), a new line (line feed).
- "\r" (ASCII *13* (*0x0D*)), a carriage return.
- "\0" (ASCII *0* (*0x00*)), the *NUL* -byte.
- "\x0B" (ASCII *11* (*0x0B*)), a vertical tab.

Parameters

`str`
: The input string.

`charlist`
: You can also specify the characters you want to strip, by means of the *`charlist`* parameter. Simply list all characters that you want to be stripped. With*..* you can specify a range of characters.

Return Values

Returns the modified string.

ChangeLog

Version	Description
4.1.0	The *`charlist`* parameter was added.

Examples

Example #34 - Usage example of rtrim()

```
<?php

$text = "\t\tThese are a few words :) ...  ";
$binary = "\x09Example string\x0A";
$hello  = "Hello World";
var_dump($text, $binary, $hello);

print "\n";

$trimmed = rtrim($text);
var_dump($trimmed);

$trimmed = rtrim($text, " \t.");
var_dump($trimmed);

$trimmed = rtrim($hello, "Hdle");
var_dump($trimmed);

// trim the ASCII control characters at the end of $binary
// (from 0 to 31 inclusive)
$clean = rtrim($binary, "\x00..\x1F");
var_dump($clean);

?>
```

The above example will output:

```
string(32) "        These are a few words :) ...  "
string(16) "    Example string
"
string(11) "Hello World"

string(30) "        These are a few words :) ..."
string(26) "        These are a few words :)"
string(9) "Hello Wor"
string(15) "    Example string"
```

See Also

- trim()
- ltrim()

setlocale

setlocale -- Set locale information

Description

string **setlocale** (int `$category`, string `$locale` [, string `$...`])

string **setlocale** (int `$category`, array `$locale`)

Sets locale information.

Parameters

`category`
`category` is a named constant specifying the category of the functions affected by the locale setting:

- LC_ALL for all of the below
- LC_COLLATE for string comparison, see strcoll()
- LC_CTYPE for character classification and conversion, for example strtoupper()
- LC_MONETARY for localeconv()
- LC_NUMERIC for decimal separator (See also localeconv())
- LC_TIME for date and time formatting with strftime()
- LC_MESSAGES for system responses (available if PHP was compiled with *libintl*)

`locale`
If `locale` is **NULL** or the empty string *""*, the locale names will be set from the values of environment variables with the same names as the above categories, or from "LANG". If `locale` is *"0"*, the locale setting is not affected, only the current setting is returned. If `locale` is an array or followed by additional parameters then each array element or parameter is tried to be set as new locale until success. This is useful if a locale is known under different names on different systems or for providing a fallback for a possibly not available locale.

`...`

Return Values

Returns the new current locale, or **FALSE** if the locale functionality is not implemented on your platform, the specified locale does not exist or the category name is invalid.

An invalid category name also causes a warning message. Category/locale names can be found in » RFC 1766 and » ISO 639. Different systems have different naming schemes for locales.

Note
The return value of setlocale() depends on the system that PHP is running. It returns exactly what the system setlocale function returns.

ChangeLog

Version	Description
4.3.0	Passing multiple locales became possible.
4.2.0	Passing *category* as a string is now deprecated, use the above constants instead. Passing them as a string (within quotes) will result in a warning message.

Examples

Example #35 - setlocale() Examples

```
<?php
/* Set locale to Dutch */
setlocale(LC_ALL, 'nl_NL');

/* Output: vrijdag 22 december 1978 */
echo strftime("%A %e %B %Y", mktime(0, 0, 0, 12, 22, 1978));

/* try different possible locale names for german as of PHP 4.3.0 */
$loc_de = setlocale(LC_ALL, 'de_DE@euro', 'de_DE', 'de', 'ge');
echo "Preferred locale for german on this system is '$loc_de'";
?>
```

Example #36 - setlocale() Examples for Windows

```
<?php
/* Set locale to Dutch */
setlocale(LC_ALL, 'nld_nld');
```

```
/* Output: vrijdag 22 december 1978 */
echo strftime("%A %d %B %Y", mktime(0, 0, 0, 12, 22, 1978));

/* try different possible locale names for german as of PHP 4.3.0 */
$loc_de = setlocale(LC_ALL, 'de_DE@euro', 'de_DE', 'deu_deu');
echo "Preferred locale for german on this system is '$loc_de'";
?>
```

Notes

Warning

The locale information is maintained per process, not per thread. If you are running PHP on a multithreaded server api like IIS or Apache on Windows you may experience sudden changes of locale settings while a script is running although the script itself never called setlocale() itself. This happens due to other scripts running in different threads of the same process at the same time changing the processwide locale using setlocale().

Tip

Windows users will find useful information about *locale* strings at Microsoft's MSDN website. Supported language strings can be found at » http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_language_strings.asp and supported country/region strings at » http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_country_strings.asp. Windows systems support the three letter codes for country/region specified by *ISO 3166-Alpha-3*, which can be found at this » Unicode website.

sha1_file

sha1_file -- Calculate the sha1 hash of a file

Description

string **sha1_file** (string `$filename` [, bool `$raw_output`])

Calculates the sha1 hash of *`filename`* using the » US Secure Hash Algorithm 1, and returns that hash. The hash is a 40-character hexadecimal number.

Parameters

`filename`
: The filename

`raw_output`
: When **TRUE**, returns the digest in raw binary format with a length of 20. Defaults to **FALSE**.

Return Values

Returns a string on success, **FALSE** otherwise.

ChangeLog

Version	Description
5.0.0	Added the *`raw_output`* parameter
5.1.0	Changed the function to use the streams API. It means that you can use it with wrappers, like *sha1_file('http://example.com/..')*

See Also

- sha1()
- md5_file()
- crc32()

sha1

sha1 -- Calculate the sha1 hash of a string

Description

string **sha1** (string `$str` [, bool `$raw_output`])

Calculates the sha1 hash of *`str`* using the » US Secure Hash Algorithm 1.

Parameters

`str`
: The input string.

`raw_output`
: If the optional *`raw_output`* is set to **TRUE**, then the sha1 digest is instead returned in raw binary format with a length of 20, otherwise the returned value is a 40-character hexadecimal number. Defaults to **FALSE**.

Return Values

Returns the sha1 hash as a string.

ChangeLog

Version	Description
5.0.0	The *`raw_output`* parameter was added.

Examples

Example #37 - A sha1() example

```
<?php
$str = 'apple';

if (sha1($str) === 'd0be2dc421be4fcd0172e5afceea3970e2f3d940') {
    echo "Would you like a green or red apple?";
    exit;
}
```

```
?>
```

See Also

- sha1_file()
- crc32()
- md5()

similar_text

similar_text -- Calculate the similarity between two strings

Description

int **similar_text** (string `$first`, string `$second` [, float `&$percent`])

This calculates the similarity between two strings as described in Oliver [1993]. Note that this implementation does not use a stack as in Oliver's pseudo code, but recursive calls which may or may not speed up the whole process. Note also that the complexity of this algorithm is O(N**3) where N is the length of the longest string.

Parameters

`first`
The first string.

`second`
The second string.

`percent`
By passing a reference as third argument, similar_text() will calculate the similarity in percent for you.

Return Values

Returns the number of matching chars in both strings.

See Also

- levenshtein()
- soundex()

soundex

soundex -- Calculate the soundex key of a string

Description

string **soundex** (string `$str`)

Calculates the soundex key of *`str`*.

Soundex keys have the property that words pronounced similarly produce the same soundex key, and can thus be used to simplify searches in databases where you know the pronunciation but not the spelling. This soundex function returns a string 4 characters long, starting with a letter.

This particular soundex function is one described by Donald Knuth in "The Art Of Computer Programming, vol. 3: Sorting And Searching", Addison-Wesley (1973), pp. 391-392.

Parameters

`str`
The input string.

Return Values

Returns the soundex key as a string.

Examples

Example #38 - Soundex Examples

```
<?php
soundex("Euler")       == soundex("Ellery");    // E460
soundex("Gauss")       == soundex("Ghosh");     // G200
soundex("Hilbert")     == soundex("Heilbronn"); // H416
soundex("Knuth")       == soundex("Kant");      // K530
soundex("Lloyd")       == soundex("Ladd");      // L300
soundex("Lukasiewicz") == soundex("Lissajous"); // L222
?>
```

See Also

- levenshtein()
- metaphone()
- similar_text()

sprintf

sprintf -- Return a formatted string

Description

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

Returns a string produced according to the formatting string *`format`*.

Parameters

`format`

The format string is composed of zero or more directives: ordinary characters (excluding *%*) that are copied directly to the result, and *conversion specifications*, each of which results in fetching its own parameter. This applies to both sprintf() and printf(). Each conversion specification consists of a percent sign (*%*), followed by one or more of these elements, in order:

- An optional *sign specifier* that forces a sign (- or +) to be used on a number. By default, only the - sign is used on a number if it's negative. This specifier forces positive numbers to have the + sign attached as well, and was added in PHP 4.3.0.
- An optional *padding specifier* that says what character will be used for padding the results to the right string size. This may be a space character or a *0* (zero character). The default is to pad with spaces. An alternate padding character can be specified by prefixing it with a single quote (*'*). See the examples below.
- An optional *alignment specifier* that says if the result should be left-justified or right-justified. The default is right-justified; a - character here will make it left-justified.
- An optional number, a *width specifier* that says how many characters (minimum) this conversion should result in.
- An optional *precision specifier* that says how many decimal digits should be displayed for floating-point numbers. When using this specifier on a string, it acts as a cutoff point, setting a maximum character limit to the string.
- A *type specifier* that says what type the argument data should be treated as. Possible types:
  - *%* - a literal percent character. No argument is required.
  - *b* - the argument is treated as an integer, and presented as a binary number.
  - *c* - the argument is treated as an integer, and presented as the character with that ASCII value.
  - *d* - the argument is treated as an integer, and presented as a (signed) decimal number.
  - *e* - the argument is treated as scientific notation (e.g. 1.2e+2). The precision specifier stands for the number of digits after the decimal point since PHP

5.2.1. In earlier versions, it was taken as number of significant digits (one less).

- *u* - the argument is treated as an integer, and presented as an unsigned decimal number.
- *f* - the argument is treated as a float, and presented as a floating-point number (locale aware).
- *F* - the argument is treated as a float, and presented as a floating-point number (non-locale aware). Available since PHP 4.3.10 and PHP 5.0.3.
- *o* - the argument is treated as an integer, and presented as an octal number.
- *s* - the argument is treated as and presented as a string.
- *x* - the argument is treated as an integer and presented as a hexadecimal number (with lowercase letters).
- *X* - the argument is treated as an integer and presented as a hexadecimal number (with uppercase letters).

The format string supports argument numbering/swapping. Here is an example:

Example #39 - Argument swapping

```
<?php
$format = 'There are %d monkeys in the %s';
printf($format, $num, $location);
?>
```

This might output, "There are 5 monkeys in the tree". But imagine we are creating a format string in a separate file, commonly because we would like to internationalize it and we rewrite it as:

Example #40 - Argument swapping

```
<?php
$format = 'The %s contains %d monkeys';
printf($format, $num, $location);
?>
```

We now have a problem. The order of the placeholders in the format string does not match the order of the arguments in the code. We would like to leave the code as is and simply indicate in the format string which arguments the placeholders refer to. We would write the format string like this instead:

Example #41 - Argument swapping

```
<?php
$format = 'The %2$s contains %1$d monkeys';
printf($format, $num, $location);
?>
```

An added benefit here is that you can repeat the placeholders without adding more arguments in the code. For example:

Example #42 - Argument swapping

```
<?php
$format = 'The %2$s contains %1$d monkeys.
           That\'s a nice %2$s full of %1$d monkeys.';
```

```
printf($format, $num, $location);
?>
```

args

...

Return Values

Returns a string produced according to the formatting string *format*.

ChangeLog

Version	Description
4.0.6	Support for argument numbering/swapping was added

Examples

Example #43 - printf(): various examples

```
<?php
$n =  43951789;
$u = -43951789;
$c = 65; // ASCII 65 is 'A'

// notice the double %%, this prints a literal '%' character
printf("%%b = '%b'\n", $n); // binary representation
printf("%%c = '%c'\n", $c); // print the ascii character, same as chr()
function
printf("%%d = '%d'\n", $n); // standard integer representation
printf("%%e = '%e'\n", $n); // scientific notation
printf("%%u = '%u'\n", $n); // unsigned integer representation of a positive
integer
printf("%%u = '%u'\n", $u); // unsigned integer representation of a negative
integer
printf("%%f = '%f'\n", $n); // floating point representation
printf("%%o = '%o'\n", $n); // octal representation
printf("%%s = '%s'\n", $n); // string representation
printf("%%x = '%x'\n", $n); // hexadecimal representation (lower-case)
printf("%%X = '%X'\n", $n); // hexadecimal representation (upper-case)

printf("%%+d = '%+d'\n", $n); // sign specifier on a positive integer
printf("%%+d = '%+d'\n", $u); // sign specifier on a negative integer
```

```
?>
```

The above example will output:

```
%b = '10100111101010011010101101'
%c = 'A'
%d = '43951789'
%e = '4.39518e+7'
%u = '43951789'
%u = '4251015507'
%f = '43951789.000000'
%o = '247523255'
%s = '43951789'
%x = '29ea6ad'
%X = '29EA6AD'
%+d = '+43951789'
%+d = '-43951789'
```

Example #44 - printf(): string specifiers

```
<?php
$s = 'monkey';
$t = 'many monkeys';

printf("[%s]\n",      $s); // standard string output
printf("[%10s]\n",    $s); // right-justification with spaces
printf("[%-10s]\n",   $s); // left-justification with spaces
printf("[%010s]\n",   $s); // zero-padding works on strings too
printf("[%'#10s]\n",  $s); // use the custom padding character '#'
printf("[%10.10s]\n", $t); // left-justification but with a cutoff of 10
characters
?>
```

The above example will output:

```
[monkey]
[    monkey]
[monkey    ]
[0000monkey]
[####monkey]
[many monke]
```

Example #45 - sprintf(): zero-padded integers

```
<?php
$isodate = sprintf("%04d-%02d-%02d", $year, $month, $day);
?>
```

Example #46 - sprintf(): formatting currency

```
<?php
$money1 = 68.75;
```

```
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money will output "123.1";
$formatted = sprintf("%01.2f", $money);
// echo $formatted will output "123.10"
?>
```

Example #47 - sprintf(): scientific notation

```
<?php
$number = 362525200;

echo sprintf("%.3e", $number); // outputs 3.625e+8
?>
```

See Also

- printf()
- sscanf()
- fscanf()
- vsprintf()
- number_format()

sscanf

sscanf -- Parses input from a string according to a format

Description

mixed **sscanf** (string `$str`, string `$format` [, mixed `&$...`])

The function sscanf() is the input analog of printf(). sscanf() reads from the string *`str`* and interprets it according to the specified *`format`*, which is described in the documentation for sprintf().

Any whitespace in the format string matches any whitespace in the input string. This means that even a tab \t in the format string can match a single space character in the input string.

Parameters

`str`
The input string being parsed.

`format`
The interpreted format for *`str`*, which is described in the documentation for sprintf().

`...`
Optionally pass in variables by reference that will contain the parsed values.

Return Values

If only two parameters were passed to this function, the values parsed will be returned as an array. Otherwise, if optional parameters are passed, the function will return the number of assigned values. The optional parameters must be passed by reference.

Examples

Example #48 - sscanf() Example

```
<?php
// getting the serial number
list($serial) = sscanf("SN/2350001", "SN/%d");
// and the date of manufacturing
$mandate = "January 01 2000";
list($month, $day, $year) = sscanf($mandate, "%s %d %d");
echo "Item $serial was manufactured on: $year-" . substr($month, 0, 3) .
"-$day\n";
?>
```

If optional parameters are passed, the function will return the number of assigned values.

Example #49 - sscanf() - using optional parameters

```
<?php
// get author info and generate DocBook entry
$auth = "24\tLewis Carroll";
$n = sscanf($auth, "%d\t%s %s", $id, $first, $last);
echo "<author id='$id'>
   <firstname>$first</firstname>
   <surname>$last</surname>
</author>\n";
?>
```

See Also

- fscanf()
- printf()
- sprintf()

str_getcsv

str_getcsv -- Parse a CSV string into an array

Description

array **str_getcsv** (string `$input` [, string `$delimiter` [, string `$enclosure` [, string `$escape`]]])

Warning
This function is currently not documented; only its argument list is available.

See Also

- fgetcsv()

str_ireplace

str_ireplace -- Case-insensitive version of str_replace().

Description

mixed **str_ireplace** (mixed `$search`, mixed `$replace`, mixed `$subject` [, int `&$count`])

This function returns a string or an array with all occurrences of *`search`* in *`subject`* (ignoring case) replaced with the given *`replace`* value. If you don't need fancy replacing rules, you should generally use this function instead of eregi_replace() or preg_replace() with the *i* modifier.

Parameters

`search`

Note
Every replacement with *`search`* array is performed on the result of previous replacement.

`replace`

`subject`
If *`subject`* is an array, then the search and replace is performed with every entry of *`subject`*, and the return value is an array as well.

`count`
The number of matched and replaced *`needles`* will be returned in *`count`* which is passed by reference.

If *`search`* and *`replace`* are arrays, then str_ireplace() takes a value from each array and uses them to do search and replace on *`subject`*. If *`replace`* has fewer values than *`search`*, then an empty string is used for the rest of replacement values. If *`search`* is an array and *`replace`* is a string; then this replacement string is used for every value of *`search`*.

Return Values

Returns a string or an array of replacements.

ChangeLog

Version	Description
5.0.0	The *count* parameter was added.

Examples

Example #50 - str_ireplace() example

```
<?php
$bodytag = str_ireplace("%body%", "black", "<body text=%BODY%>");
?>
```

Notes

Note
This function is binary safe.

See Also

- str_replace()
- preg_replace()
- strtr()

str_pad

str_pad -- Pad a string to a certain length with another string

Description

string **str_pad** (string `$input`, int `$pad_length` [, string `$pad_string` [, int `$pad_type`]])

This functions returns the *`input`* string padded on the left, the right, or both sides to the specified padding length. If the optional argument *`pad_string`* is not supplied, the *`input`* is padded with spaces, otherwise it is padded with characters from *`pad_string`* up to the limit.

Parameters

`input`
The input string.

`pad_length`
If the value of *`pad_length`* is negative or less than the length of the input string, no padding takes place.

`pad_string`

Note
The *`pad_string`* may be truncated if the required number of padding characters can't be evenly divided by the *`pad_string`* 's length.

`pad_type`
Optional argument *`pad_type`* can be **STR_PAD_RIGHT**, **STR_PAD_LEFT**, or **STR_PAD_BOTH**. If *`pad_type`* is not specified it is assumed to be **STR_PAD_RIGHT**.

Return Values

Returns the padded string.

Examples

Example #51 - str_pad() example

```
<?php
$input = "Alien";
echo str_pad($input, 10);                      // produces "Alien     "
echo str_pad($input, 10, "-=", STR_PAD_LEFT);  // produces "-=-=-Alien"
echo str_pad($input, 10, "_", STR_PAD_BOTH);   // produces "__Alien___"
echo str_pad($input, 6 , "___");               // produces "Alien_"
?>
```

str_repeat

str_repeat -- Repeat a string

Description

string **str_repeat** (string `$input`, int `$multiplier`)

Returns *`input`* repeated *`multiplier`* times.

Parameters

`input`
: The string to be repeated.

`multiplier`
: Number of time the *`input`* string should be repeated. *`multiplier`* has to be greater than or equal to 0. If the *`multiplier`* is set to 0, the function will return an empty string.

Return Values

Returns the repeated string.

Examples

Example #52 - str_repeat() example

```
<?php
echo str_repeat("-=", 10);
?>
```

The above example will output:

```
-=-=-=-=-=-=-=-=-=-=
```

See Also

- for
- str_pad()
- substr_count()

str_replace

str_replace -- Replace all occurrences of the search string with the replacement string

Description

mixed **str_replace** (mixed `$search`, mixed `$replace`, mixed `$subject` [, int `&$count`])

This function returns a string or an array with all occurrences of *`search`* in *`subject`* replaced with the given *`replace`* value.

If you don't need fancy replacing rules (like regular expressions), you should always use this function instead of ereg_replace() or preg_replace().

Parameters

If *`search`* and *`replace`* are arrays, then str_replace() takes a value from each array and uses them to do search and replace on *`subject`*. If *`replace`* has fewer values than *`search`*, then an empty string is used for the rest of replacement values. If *`search`* is an array and *`replace`* is a string, then this replacement string is used for every value of *`search`*. The converse would not make sense, though.

If *`search`* or *`replace`* are arrays, their elements are processed first to last.

`search`

`replace`

`subject`
If *`subject`* is an array, then the search and replace is performed with every entry of *`subject`*, and the return value is an array as well.

`count`

Note
If passed, this will hold the number of matched and replaced needles.

Return Values

This function returns a string or an array with the replaced values.

ChangeLog

Version	Description
5.0.0	The *count* parameter was added.
4.3.3	The behaviour of this function changed. In older versions a bug existed when using arrays as both *search* and *replace* parameters which caused empty *search* indexes to be skipped without advancing the internal pointer on the *replace* array. This has been corrected in PHP 4.3.3, any scripts which relied on this bug should remove empty search values prior to calling this function in order to mimic the original behavior.
4.0.5	Most parameters can now be an array.

Examples

Example #53 - str_replace() examples

```
<?php
// Provides: <body text='black'>
$bodytag = str_replace("%body%", "black", "<body text='%body%'>");

// Provides: Hll Wrld f PHP
$vowels = array("a", "e", "i", "o", "u", "A", "E", "I", "O", "U");
$onlyconsonants = str_replace($vowels, "", "Hello World of PHP");

// Provides: You should eat pizza, beer, and ice cream every day
$phrase  = "You should eat fruits, vegetables, and fiber every day.";
$healthy = array("fruits", "vegetables", "fiber");
$yummy   = array("pizza", "beer", "ice cream");

$newphrase = str_replace($healthy, $yummy, $phrase);

// Use of the count parameter is available as of PHP 5.0.0
$str = str_replace("ll", "", "good golly miss molly!", $count);
echo $count; // 2

// Order of replacement
$str     = "Line 1\nLine 2\rLine 3\r\nLine 4\n";
$order   = array("\r\n", "\n", "\r");
$replace = '<br />';
// Processes \r\n's first so they aren't converted twice.
$newstr = str_replace($order, $replace, $str);
```

```
// Outputs: apearpearle pear
$letters = array('a', 'p');
$fruit   = array('apple', 'pear');
$text    = 'a p';
$output  = str_replace($letters, $fruit, $text);
echo $output;
?>
```

Notes

Note
This function is binary-safe.

Note
This function is case-sensitive. Use str_ireplace() for case-insensitive replace.

See Also

- str_ireplace()
- substr_replace()
- preg_replace()
- strtr()

str_rot13

str_rot13 -- Perform the rot13 transform on a string

Description

string **str_rot13** (string `$str`)

Performs the ROT13 encoding on the *`str`* argument and returns the resulting string.

The ROT13 encoding simply shifts every letter by 13 places in the alphabet while leaving non-alpha characters untouched. Encoding and decoding are done by the same function, passing an encoded string as argument will return the original version.

Parameters

`str`
The input string.

Return Values

Returns the ROT13 version of the given string.

Examples

Example #54 - str_rot13() example

```
<?php

echo str_rot13('PHP 4.3.0'); // CUC 4.3.0

?>
```

ChangeLog

Version	Description
4.3.0	The behaviour of this function was fixed. Before this fix, the *`str`* was also modified, as if it was passed by reference.

str_shuffle

str_shuffle -- Randomly shuffles a string

Description

string **str_shuffle** (string `$str`)

str_shuffle() shuffles a string. One permutation of all possible is created.

Parameters

`str`
The input string.

Return Values

Returns the shuffled string.

Examples

Example #55 - str_shuffle() example

```
<?php
$str = 'abcdef';
$shuffled = str_shuffle($str);

// This will echo something like: bfdaec
echo $shuffled;
?>
```

See Also

- shuffle()
- rand()

str_split

str_split -- Convert a string to an array

Description

array **str_split** (string `$string` [, int `$split_length`])

Converts a string to an array.

Parameters

`string`
The input string.

`split_length`
Maximum length of the chunk.

Return Values

If the optional *`split_length`* parameter is specified, the returned array will be broken down into chunks with each being *`split_length`* in length, otherwise each chunk will be one character in length.

FALSE is returned if *`split_length`* is less than 1. If the *`split_length`* length exceeds the length of *`string`*, the entire string is returned as the first (and only) array element.

Examples

Example #56 - Example uses of str_split()

```
<?php

$str = "Hello Friend";

$arr1 = str_split($str);
$arr2 = str_split($str, 3);

print_r($arr1);
print_r($arr2);

?>
```

The above example will output:

```
Array
```

```
(
    [0] => H
    [1] => e
    [2] => l
    [3] => l
    [4] => o
    [5] =>
    [6] => F
    [7] => r
    [8] => i
    [9] => e
    [10] => n
    [11] => d
)

Array
(
    [0] => Hel
    [1] => lo
    [2] => Fri
    [3] => end
)
```

See Also

- chunk_split()
- preg_split()
- explode()
- count_chars()
- str_word_count()
- for

str_word_count

str_word_count -- Return information about words used in a string

Description

mixed **str_word_count** (string `$string` [, int `$format` [, string `$charlist`]])

Counts the number of words inside *`string`*. If the optional *`format`* is not specified, then the return value will be an integer representing the number of words found. In the event the *`format`* is specified, the return value will be an array, content of which is dependent on the *`format`*. The possible value for the *`format`* and the resultant outputs are listed below.

For the purpose of this function, 'word' is defined as a locale dependent string containing alphabetic characters, which also may contain, but not start with "'" and "-" characters.

Parameters

`string`
The string

`format`
Specify the return value of this function. The current supported values are:

- 0 - returns the number of words found
- 1 - returns an array containing all the words found inside the *`string`*
- 2 - returns an associative array, where the key is the numeric position of the word inside the *`string`* and the value is the actual word itself

`charlist`
A list of additional characters which will be considered as 'word'

Return Values

Returns an array or an integer, depending on the *`format`* chosen.

ChangeLog

Version	Description
5.1.0	Added the *`charlist`* parameter

Examples

Example #57 - A str_word_count() example

```
<?php

$str = "Hello fri3nd, you're
      looking          good today!";

print_r(str_word_count($str, 1));
print_r(str_word_count($str, 2));
print_r(str_word_count($str, 1, 'àáãç3'));

echo str_word_count($str);

?>
```

The above example will output:

```
Array
(
   [0] => Hello
   [1] => fri
   [2] => nd
   [3] => you're
   [4] => looking
   [5] => good
   [6] => today
)

Array
(
   [0] => Hello
   [6] => fri
   [10] => nd
   [14] => you're
   [29] => looking
   [46] => good
   [51] => today
)

Array
(
   [0] => Hello
   [1] => fri3nd
   [2] => you're
   [3] => looking
   [4] => good
   [5] => today
)

7
```

See Also

- explode()
- preg_split()
- split()
- count_chars()
- substr_count()

strcasecmp

strcasecmp -- Binary safe case-insensitive string comparison

Description

int **strcasecmp** (string `$str1`, string `$str2`)

Binary safe case-insensitive string comparison.

Parameters

`str1`
The first string

`str2`
The second string

Return Values

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

Examples

Example #58 - strcasecmp() example

```
<?php
$var1 = "Hello";
$var2 = "hello";
if (strcasecmp($var1, $var2) == 0) {
   echo '$var1 is equal to $var2 in a case-insensitive string comparison';
}
?>
```

See Also

- preg_match()
- strcmp()
- substr()
- stristr()

- strncasecmp()
- strstr()

strchr

strchr -- Alias of strstr()

Description

This function is an alias of: strstr().

strcmp

strcmp -- Binary safe string comparison

Description

int **strcmp** (string `$str1`, string `$str2`)

Note that this comparison is case sensitive.

Parameters

`str1`
: The first string.

`str2`
: The second string.

Return Values

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

See Also

- preg_match()
- strcasecmp()
- substr()
- stristr()
- strncasecmp()
- strncmp()
- strstr()

strcoll

strcoll -- Locale based string comparison

Description

int **strcoll** (string `$str1`, string `$str2`)

Note that this comparison is case sensitive, and unlike strcmp() this function is not binary safe.

strcoll() uses the current locale for doing the comparisons. If the current locale is C or POSIX, this function is equivalent to strcmp().

Parameters

`str1`
The first string.

`str2`
The second string.

Return Values

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

ChangeLog

Version	Description
4.2.3	This function now works on win32.

See Also

- preg_match()
- strcmp()
- strcasecmp()
- substr()
- stristr()

- strncasecmp()
- strncmp()
- strstr()
- setlocale()

strcspn

strcspn -- Find length of initial segment not matching mask

Description

int **strcspn** (string `$str1`, string `$str2` [, int `$start` [, int `$length`]])

Returns the length of the initial segment of *`str1`* which does *not* contain any of the characters in *`str2`*.

Parameters

`str1`
: The first string.

`str2`
: The second string.

`start`
: The start position of the string to examine.

`length`
: The length of the string to examine.

Return Values

Returns the length of the segment as an integer.

ChangeLog

Version	Description
4.3.0	The *`start`* and *`length`* were added

Notes

Note
This function is binary-safe.

See Also

- strspn()

strip_tags

strip_tags -- Strip HTML and PHP tags from a string

Description

string **strip_tags** (string `$str` [, string `$allowable_tags`])

This function tries to return a string with all HTML and PHP tags stripped from a given *`str`*. It uses the same tag stripping state machine as the fgetss() function.

Parameters

`str`
: The input string.

`allowable_tags`
: You can use the optional second parameter to specify tags which should not be stripped.

Note
HTML comments and PHP tags are also stripped. This is hardcoded and can not be changed with *`allowable_tags`*.

Return Values

Returns the stripped string.

ChangeLog

Version	Description
5.0.0	strip_tags() is now binary safe
4.3.0	HTML comments are now always stripped
4.0.0	The *`allowable_tags`* parameter was added

Examples

Example #59 - strip_tags() example

```
<?php
$text = '<p>Test paragraph.</p><!-- Comment --> <a href="#fragment">Other
text</a>';
echo strip_tags($text);
echo "\n";

// Allow <p> and <a>
echo strip_tags($text, '<p><a>');
?>
```

The above example will output:

```
Test paragraph. Other text
<p>Test paragraph.</p> <a href="#fragment">Other text</a>
```

Notes

Warning

Because strip_tags() does not actually validate the HTML, partial, or broken tags can result in the removal of more text/data than expected.

Warning

This function does not modify any attributes on the tags that you allow using `allowable_tags`, including the *style* and *onmouseover* attributes that a mischievous user may abuse when posting text that will be shown to other users.

See Also

- htmlspecialchars()

stripcslashes

stripcslashes -- Un-quote string quoted with addcslashes()

Description

string **stripcslashes** (string `$str`)

Returns a string with backslashes stripped off. Recognizes C-like *n*, *r*..., octal and hexadecimal representation.

Parameters

`str`
: The string to be unescaped.

Return Values

Returns the unescaped string.

See Also

- addcslashes()

stripos

stripos -- Find position of first occurrence of a case-insensitive string

Description

int **stripos** (string `$haystack`, string `$needle` [, int `$offset`])

Returns the numeric position of the first occurrence of *`needle`* in the *`haystack`* string.

Unlike strpos(), stripos() is case-insensitive.

Parameters

`haystack`
The string to search in

`needle`
Note that the *`needle`* may be a string of one or more characters. If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

`offset`
The optional *`offset`* parameter allows you to specify which character in *`haystack`* to start searching. The position returned is still relative to the beginning of *`haystack`*.

Return Values

If *`needle`* is not found, stripos() will return boolean **FALSE**.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #60 - stripos() examples

```
<?php
$findme    = 'a';
$mystring1 = 'xyz';
$mystring2 = 'ABC';
```

```
$pos1 = stripos($mystring1, $findme);
$pos2 = stripos($mystring2, $findme);

// Nope, 'a' is certainly not in 'xyz'
if ($pos1 === false) {
   echo "The string '$findme' was not found in the string '$mystring1'";
}

// Note our use of ===.  Simply == would not work as expected
// because the position of 'a' is the 0th (first) character.
if ($pos2 !== false) {
   echo "We found '$findme' in '$mystring2' at position $pos2";
}
?>
```

Notes

Note
This function is binary-safe.

See Also

- strpos()
- strrpos()
- strrchr()
- substr()
- stristr()
- strstr()
- strripos()
- str_ireplace()

stripslashes

stripslashes -- Un-quote string quoted with addslashes()

Description

string **stripslashes** (string `$str`)

Un-quotes a quoted string.

Note
If magic_quotes_sybase is on, no backslashes are stripped off but two apostrophes are replaced by one instead.

An example use of stripslashes() is when the PHP directive magic_quotes_gpc is *on* (it's on by default), and you aren't inserting this data into a place (such as a database) that requires escaping. For example, if you're simply outputting data straight from an HTML form.

Parameters

`str`
The input string.

Return Values

Returns a string with backslashes stripped off. (\' becomes ' and so on.) Double backslashes (\\ ) are made into a single backslash (\ ).

Examples

Example #61 - A stripslashes() example

```
<?php
$str = "Is your name O\'reilly?";

// Outputs: Is your name O'reilly?
echo stripslashes($str);
?>
```

Note

stripslashes() is not recursive. If you want to apply this function to a multi-dimensional array, you need to use a recursive function.

Example #62 - Using stripslashes() on an array

```
<?php
function stripslashes_deep($value)
{
   $value = is_array($value) ?
               array_map('stripslashes_deep', $value) :
               stripslashes($value);

   return $value;
}

// Example
$array = array("f\\'oo", "b\\'ar", array("fo\\'o", "b\\'ar"));
$array = stripslashes_deep($array);

// Output
print_r($array);
?>
```

The above example will output:

```
Array
(
   [0] => f'oo
   [1] => b'ar
   [2] => Array
       (
           [0] => fo'o
           [1] => b'ar
       )

)
```

See Also

- addslashes()
- get_magic_quotes_gpc()

stristr

stristr -- Case-insensitive strstr()

Description

string **stristr** (string `$haystack`, mixed `$needle` [, bool `$before_needle`])

Returns all of *`haystack`* from the first occurrence of *`needle`* to the end.

Parameters

`haystack`
The string to search in

`needle`
If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

`before_needle`
If **TRUE** (the default is **FALSE**), stristr() returns the part of the *`haystack`* before the first occurence of the *`needle`*.

`needle` and *`haystack`* are examined in a case-insensitive manner.

Return Values

Returns the matched substring. If *`needle`* is not found, returns **FALSE**.

ChangeLog

Version	Description
5.3.0	Added the optional parameter *`before_needle`*.
4.3.0	stristr() was made binary safe.

Examples

Example #63 - stristr() example

```
<?php
 $email = 'USER@EXAMPLE.com';
 echo stristr($email, 'e'); // outputs ER@EXAMPLE.com
 echo stristr($email, 'e', true); // As of PHP 5.3.0, outputs US
?>
```

Example #64 - Testing if a string is found or not

```
<?php
 $string = 'Hello World!';
 if(stristr($string, 'earth') === FALSE) {
   echo '"earth" not found in string';
 }
// outputs: "earth" not found in string
?>
```

Example #65 - Using a non "string" needle

```
<?php
 $string = 'APPLE';
 echo stristr($string, 97); // 97 = lowercase a
// outputs: APPLE
?>
```

Notes

Note

This function is binary-safe.

See Also

- strstr()
- strrchr()
- substr()
- preg_match()

strlen

strlen -- Get string length

Description

int **strlen** (string `$string`)

Returns the length of the given *`string`*.

Parameters

`string`
The string being measured for length.

Return Values

The length of the *`string`* on success, and *0* if the *`string`* is empty.

Examples

Example #66 - A strlen() example

```
<?php
$str = 'abcdef';
echo strlen($str); // 6

$str = ' ab cd ';
echo strlen($str); // 7
?>
```

See Also

- count()
- mb_strlen()

strnatcasecmp

strnatcasecmp -- Case insensitive string comparisons using a "natural order" algorithm

Description

int **strnatcasecmp** (string `$str1`, string `$str2`)

This function implements a comparison algorithm that orders alphanumeric strings in the way a human being would. The behaviour of this function is similar to strnatcmp(), except that the comparison is not case sensitive. For more information see: Martin Pool's » Natural Order String Comparison page.

Parameters

`str1`
The first string.

`str2`
The second string.

Return Values

Similar to other string comparison functions, this one returns < 0 if *`str1`* is less than *`str2`* > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

See Also

- preg_match()
- strcmp()
- strcasecmp()
- substr()
- stristr()
- strncasecmp()
- strncmp()
- strstr()
- setlocale()

strnatcmp

strnatcmp -- String comparisons using a "natural order" algorithm

Description

int **strnatcmp** (string `$str1`, string `$str2`)

This function implements a comparison algorithm that orders alphanumeric strings in the way a human being would, this is described as a "natural ordering". Note that this comparison is case sensitive.

Parameters

`str1`
The first string.

`str2`
The second string.

Return Values

Similar to other string comparison functions, this one returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

Examples

An example of the difference between this algorithm and the regular computer string sorting algorithms (used in strcmp()) can be seen below:

```
<?php
$arr1 = $arr2 = array("img12.png", "img10.png", "img2.png", "img1.png");
echo "Standard string comparison\n";
usort($arr1, "strcmp");
print_r($arr1);
echo "\nNatural order string comparison\n";
usort($arr2, "strnatcmp");
print_r($arr2);
?>
```

The above example will output:

```
Standard string comparison
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
```

```
)

Natural order string comparison
Array
(
   [0] => img1.png
   [1] => img2.png
   [2] => img10.png
   [3] => img12.png
)
```

For more information see: Martin Pool's » Natural Order String Comparison page.

See Also

- preg_match()
- strcasecmp()
- substr()
- stristr()
- strcmp()
- strncmp()
- strncasecmp()
- strnatcasecmp()
- strstr()
- natsort()
- natcasesort()

strncasecmp

strncasecmp -- Binary safe case-insensitive string comparison of the first n characters

Description

int **strncasecmp** (string `$str1`, string `$str2`, int `$len`)

This function is similar to strcasecmp(), with the difference that you can specify the (upper limit of the) number of characters from each string to be used in the comparison.

Parameters

`str1`
The first string.

`str2`
The second string.

`len`
The length of strings to be used in the comparison.

Return Values

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

See Also

- preg_match()
- strcmp()
- strcasecmp()
- substr()
- stristr()
- strstr()

strncmp

strncmp -- Binary safe string comparison of the first n characters

Description

int **strncmp** (string `$str1`, string `$str2`, int `$len`)

This function is similar to strcmp(), with the difference that you can specify the (upper limit of the) number of characters from each string to be used in the comparison.

Note that this comparison is case sensitive.

Parameters

`str1`
: The first string.

`str2`
: The second string.

`len`
: Number of characters to use in the comparison.

Return Values

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

See Also

- preg_match()
- strcmp()
- strcasecmp()
- substr()
- stristr()
- strncasecmp()
- strstr()

strpbrk

strpbrk -- Search a string for any of a set of characters

Description

string **strpbrk** (string `$haystack`, string `$char_list`)

strpbrk() searches the *`haystack`* string for a *`char_list`*.

Parameters

`haystack`
The string where *`char_list`* is looked for.

`char_list`
This parameter is case sensitive.

Return Values

Returns a string starting from the character found, or **FALSE** if it is not found.

Examples

Example #67 - strpbrk() example

```
<?php

$text = 'This is a Simple text.';

// this echoes "is is a Simple text." because 'i' is matched first
echo strpbrk($text, 'mi');

// this echoes "Simple text." because chars are case sensitive
echo strpbrk($text, 'S');
?>
```

strpos

strpos -- Find position of first occurrence of a string

Description

int **strpos** (string `$haystack`, mixed `$needle` [, int `$offset`])

Returns the numeric position of the first occurrence of *`needle`* in the *`haystack`* string. Unlike the strrpos() before PHP 5, this function can take a full string as the *`needle`* parameter and the entire string will be used.

Parameters

`haystack`
The string to search in

`needle`
If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

`offset`
The optional *`offset`* parameter allows you to specify which character in *`haystack`* to start searching. The position returned is still relative to the beginning of *`haystack`*.

Return Values

Returns the position as an integer. If *`needle`* is not found, strpos() will return boolean **FALSE**.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #68 - strpos() examples

```
<?php
$mystring = 'abc';
$findme   = 'a';
```

```
$pos = strpos($mystring, $findme);

// Note our use of ===.  Simply == would not work as expected
// because the position of 'a' was the 0th (first) character.
if ($pos === false) {
   echo "The string '$findme' was not found in the string '$mystring'";
} else {
   echo "The string '$findme' was found in the string '$mystring'";
   echo " and exists at position $pos";
}

// We can search for the character, ignoring anything before the offset
$newstring = 'abcdef abcdef';
$pos = strpos($newstring, 'a', 1); // $pos = 7, not 0
?>
```

Notes

Note
This function is binary-safe.

See Also

- strrpos()
- stripos()
- strripos()
- strrchr()
- substr()
- stristr()
- strstr()

strrchr

strrchr -- Find the last occurrence of a character in a string

Description

string **strrchr** (string `$haystack`, mixed `$needle`)

This function returns the portion of *`haystack`* which starts at the last occurrence of *`needle`* and goes until the end of *`haystack`*.

Parameters

`haystack`
The string to search in

`needle`
If *`needle`* contains more than one character, only the first is used. This behavior is different from that of strstr(). If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Return Values

This function returns the portion of string, or **FALSE** if *`needle`* is not found.

ChangeLog

Version	Description
4.3.0	This function is now binary safe.

Examples

Example #69 - strrchr() example

```
<?php
// get last directory in $PATH
$dir = substr(strrchr($PATH, ":"), 1);

// get everything after last newline
$text = "Line 1\nLine 2\nLine 3";
```

```
$last = substr(strrchr($text, 10), 1 );
?>
```

Notes

Note
This function is binary-safe.

See Also

- strstr()
- substr()
- stristr()

strrev

strrev -- Reverse a string

Description

string **strrev** (string `$string`)

Returns *`string`*, reversed.

Parameters

`string`
The string to be reversed.

Return Values

Returns the reversed string.

Examples

Example #70 - Reversing a string with strrev()

```
<?php
echo strrev("Hello world!"); // outputs "!dlrow olleH"
?>
```

strripos

strripos -- Find position of last occurrence of a case-insensitive string in a string

Description

int **strripos** (string `$haystack`, string `$needle` [, int `$offset`])

Find position of last occurrence of a case-insensitive string in a string. Unlike strrpos(), strripos() is case-insensitive.

Parameters

`haystack`
The string to search in

`needle`
Note that the *`needle`* may be a string of one or more characters.

`offset`
The *`offset`* parameter may be specified to begin searching an arbitrary number of characters into the string. Negative offset values will start the search at *`offset`* characters from the *start* of the string.

Return Values

Returns the numerical position of the last occurence of *`needle`*. Also note that string positions start at 0, and not 1.

If *`needle`* is not found, **FALSE** is returned.

Warning

This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #71 - A simple strripos() example

```
<?php
$haystack = 'ababcd';
```

```
$needle   = 'aB';

$pos      = strripos($haystack, $needle);

if ($pos === false) {
   echo "Sorry, we did not find ($needle) in ($haystack)";
} else {
   echo "Congratulations!\n";
   echo "We found the last ($needle) in ($haystack) at position ($pos)";
}
?>
```

The above example will output:

```
Congratulations!
  We found the last (aB) in (ababcd) at position (2)
```

See Also

- strpos()
- stripos()
- strrchr()
- substr()
- stristr()
- strstr()

strrpos

strrpos -- Find position of last occurrence of a char in a string

Description

int **strrpos** (string `$haystack`, string `$needle` [, int `$offset`])

Returns the numeric position of the last occurrence of *`needle`* in the *`haystack`* string. Note that the needle in this case can only be a single character in PHP 4. If a string is passed as the needle, then only the first character of that string will be used.

If *`needle`* is not found, returns **FALSE**.

It is easy to mistake the return values for "character found at position 0" and "character not found". Here's how to detect the difference:

```
<?php

// in PHP 4.0.0 and newer:
$pos = strrpos($mystring, "b");
if ($pos === false) { // note: three equal signs
   // not found...
}

// in versions older than 4.0.0:
$pos = strrpos($mystring, "b");
if (is_bool($pos) && !$pos) {
   // not found...
}
?>
```

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Note
As of PHP 5.0.0 *`offset`* may be specified to begin searching an arbitrary number of characters into the string. Negative values will stop searching at an arbitrary point prior to the end of the string.

Note
The *`needle`* may be a string of more than one character as of PHP 5.0.0.

Parameters

haystack

needle

offset

Return Values

See Also

- strpos()
- strripos()
- strrchr()
- substr()
- stristr()
- strstr()

strspn

strspn -- Find length of initial segment matching mask

Description

int **strspn** (string `$str1`, string `$str2` [, int `$start` [, int `$length`]])

Finds the length of the initial segment matching mask.

The line of code:

```
<?php
$var = strspn("42 is the answer, what is the question ...", "1234567890");
?>
```

will assign 2 to *$var*, because the string "42" will be the longest segment containing characters from "1234567890".

Parameters

`str1`
: The first string.

`str2`
: The second string.

`start`
: The start position of the string to examine. Negative value counts position from the end of a string.

`length`
: The length of the string to examine. Negative value sets length from the end of a string.

Return Values

Returns the length of the initial segment of *`str1`* which consists entirely of characters in *`str2`*.

ChangeLog

Version	Description
4.3.0	The *`start`* and *`length`* parameters were

	added

Examples

Example #72 - strspn() example

```
<?php
echo strspn("foo", "o", 1, 2); // 2
?>
```

Notes

Note

This function is binary-safe.

See Also

- strcspn()

strstr

strstr -- Find first occurrence of a string

Description

string **strstr** (string `$haystack`, mixed `$needle` [, bool `$before_needle`])

Returns part of *`haystack`* string from the first occurrence of *`needle`* to the end of *`haystack`*.

Note
This function is case-sensitive. For case-insensitive searches, use stristr().

Note
If you only want to determine if a particular *`needle`* occurs within *`haystack`*, use the faster and less memory intensive function strpos() instead.

Parameters

`haystack`
: The input string.

`needle`
: If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

`before_needle`
: If **TRUE** (the default is **FALSE**), strstr() returns the part of the *`haystack`* before the first occurence of the *`needle`*.

Return Values

Returns the portion of string, or **FALSE** if *`needle`* is not found.

ChangeLog

Version	Description

5.3.0	Added the optional parameter *before_needle*.
4.3.0	strstr() was made binary safe.

Examples

Example #73 - strstr() example

```
<?php
$email  = 'name@example.com';
$domain = strstr($email, '@');
echo $domain; // prints @example.com

$user = strstr($email, '@', true); // As of PHP 5.3.0
echo $user; // prints name
?>
```

See Also

- preg_match()
- stristr()
- strpos()
- strrchr()
- substr()

strtok

strtok -- Tokenize string

Description

string **strtok** (string `$str`, string `$token`)

strtok() splits a string (*`str`*) into smaller strings (tokens), with each token being delimited by any character from *`token`*. That is, if you have a string like "This is an example string" you could tokenize this string into its individual words by using the space character as the token.

Note that only the first call to strtok uses the string argument. Every subsequent call to strtok only needs the token to use, as it keeps track of where it is in the current string. To start over, or to tokenize a new string you simply call strtok with the string argument again to initialize it. Note that you may put multiple tokens in the token parameter. The string will be tokenized when any one of the characters in the argument are found.

Parameters

`str`
The string being split up into smaller strings (tokens).

`token`
The delimiter used when splitting up *`str`*.

Return Values

A string token.

Examples

Example #74 - strtok() example

```
<?php
$string = "This is\tan example\nstring";
/* Use tab and newline as tokenizing characters as well  */
$tok = strtok($string, " \n\t");

while ($tok !== false) {
   echo "Word=$tok<br />";
   $tok = strtok(" \n\t");
}
?>
```

The behavior when an empty part was found changed with PHP 4.1.0. The old behavior returned an empty string, while the new, correct, behavior simply skips the part of the string:

Example #75 - Old strtok() behavior

```
<?php
$first_token  = strtok('/something', '/');
$second_token = strtok('/');
var_dump($first_token, $second_token);
?>
```

The above example will output:

```
string(0) ""
   string(9) "something"
```

Example #76 - New strtok() behavior

```
<?php
$first_token  = strtok('/something', '/');
$second_token = strtok('/');
var_dump($first_token, $second_token);
?>
```

The above example will output:

```
string(9) "something"
   bool(false)
```

Notes

Warning

This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

See Also

- split()
- explode()

strtolower

strtolower -- Make a string lowercase

Description

string **strtolower** (string `$str`)

Returns *`string`* with all alphabetic characters converted to lowercase.

Note that 'alphabetic' is determined by the current locale. This means that in i.e. the default "C" locale, characters such as umlaut-A (Ä) will not be converted.

Parameters

`str`
The input string.

Return Values

Returns the lowercased string.

Examples

Example #77 - strtolower() example

```
<?php
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtolower($str);
echo $str; // Prints mary had a little lamb and she loved it so
?>
```

Notes

Note

This function is binary-safe.

See Also

- strtoupper()
- ucfirst()
- ucwords()
- mb_strtolower()

strtoupper

strtoupper -- Make a string uppercase

Description

string **strtoupper** (string `$string`)

Returns *`string`* with all alphabetic characters converted to uppercase.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

Parameters

`string`
The input string.

Return Values

Returns the uppercased string.

Examples

Example #78 - strtoupper() example

```
<?php
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtoupper($str);
echo $str; // Prints MARY HAD A LITTLE LAMB AND SHE LOVED IT SO
?>
```

Notes

Note

This function is binary-safe.

See Also

- strtolower()
- ucfirst()
- ucwords()
- mb_strtoupper()

strtr

strtr -- Translate certain characters

Description

string **strtr** (string `$str`, string `$from`, string `$to`)

string **strtr** (string `$str`, array `$replace_pairs`)

This function returns a copy of *`str`*, translating all occurrences of each character in *`from`* to the corresponding character in *`to`*.

If *`from`* and *`to`* are different lengths, the extra characters in the longer of the two are ignored.

Parameters

`str`
: The string being translated.

`from`
: The string being translated to *`to`*.

`to`
: The string replacing *`from`*.

`replace_pairs`
: The *`replace_pairs`* parameter may be used as a substitute for *`to`* and *`from`* in which case it's an array in the form *array('from' => 'to', ...)*.

Return Values

This function returns a copy of *`str`*, translating all occurrences of each character in *`from`* to the corresponding character in *`to`*.

ChangeLog

Version	Description
4.0.0	The optional *`to`* and *`from`* parameters were added.

Examples

Example #79 - strtr() example

```
<?php
$addr = strtr($addr, "äåö", "aao");
?>
```

strtr() may be called with only two arguments. If called with two arguments it behaves in a new way: *from* then has to be an array that contains string -> string pairs that will be replaced in the source string. strtr() will always look for the longest possible match first and will *NOT* try to replace stuff that it has already worked on.

Example #80 - strtr() example with two arguments

```
<?php
$trans = array("hello" => "hi", "hi" => "hello");
echo strtr("hi all, I said hello", $trans);
?>
```

The above example will output:

```
hello all, I said hi
```

See Also

- ereg_replace()

substr_compare

substr_compare -- Binary safe comparison of 2 strings from an offset, up to length characters

Description

int **substr_compare** (string `$main_str`, string `$str`, int `$offset` [, int `$length` [, bool `$case_insensitivity`]])

substr_compare() compares *`main_str`* from position *`offset`* with *`str`* up to *`length`* characters.

Parameters

`main_str`

`str`

`offset`
: The start position for the comparison. If negative, it starts counting from the end of the string.

`length`
: The length of the comparison.

`case_insensitivity`
: If *`case_insensitivity`* is **TRUE**, comparison is case insensitive.

Return Values

Returns < 0 if *`main_str`* from position *`offset`* is less than *`str`*, > 0 if it is greater than *`str`*, and 0 if they are equal. If *`length`* is equal or greater than length of *`main_str`* and *`length`* is set, substr_compare() prints warning and returns **FALSE**.

ChangeLog

Version	Description
5.1.0	Added the possibility to use a negative *`offset`*.

Examples

Example #81 - A substr_compare() example

```
<?php
echo substr_compare("abcde", "bc", 1, 2); // 0
echo substr_compare("abcde", "de", -2, 2); // 0
echo substr_compare("abcde", "bcg", 1, 2); // 0
echo substr_compare("abcde", "BC", 1, 2, true); // 0
echo substr_compare("abcde", "bc", 1, 3); // 1
echo substr_compare("abcde", "cd", 1, 2); // -1
echo substr_compare("abcde", "abc", 5, 1); // warning
?>
```

substr_count

substr_count -- Count the number of substring occurrences

Description

int **substr_count** (string `$haystack`, string `$needle` [, int `$offset` [, int `$length`]])

substr_count() returns the number of times the *`needle`* substring occurs in the *`haystack`* string. Please note that *`needle`* is case sensitive.

Note
This function doesn't count overlapped substrings. See the example below!

Parameters

`haystack`
: The string to search in

`needle`
: The substring to search for

`offset`
: The offset where to start counting

`length`
: The maximum length after the specified offset to search for the substring. It outputs a warning if the offset plus the length is greater than the *`haystack`* length.

Return Values

This functions returns an integer.

ChangeLog

Version	Description
5.1.0	Added the *`offset`* and the *`length`* parameters

Examples

Example #82 - A substr_count() example

```
<?php
$text = 'This is a test';
echo strlen($text); // 14

echo substr_count($text, 'is'); // 2

// the string is reduced to 's is a test', so it prints 1
echo substr_count($text, 'is', 3);

// the text is reduced to 's i', so it prints 0
echo substr_count($text, 'is', 3, 3);

// generates a warning because 5+10 > 14
echo substr_count($text, 'is', 5, 10);


// prints only 1, because it doesn't count overlapped subtrings
$text2 = 'gcdgcdgcd';
echo substr_count($text2, 'gcdgcd');
?>
```

See Also

- count_chars()
- strpos()
- substr()
- strstr()

substr_replace

substr_replace -- Replace text within a portion of a string

Description

mixed **substr_replace** (mixed `$string`, string `$replacement`, int `$start` [, int `$length`])

substr_replace() replaces a copy of *`string`* delimited by the *`start`* and (optionally) *`length`* parameters with the string given in *`replacement`*.

Parameters

`string`
The input string.

`replacement`
The replacement string.

`start`
If *`start`* is positive, the replacing will begin at the *`start`* 'th offset into *`string`*. If *`start`* is negative, the replacing will begin at the *`start`* 'th character from the end of *`string`*.

`length`
If given and is positive, it represents the length of the portion of *`string`* which is to be replaced. If it is negative, it represents the number of characters from the end of *`string`* at which to stop replacing. If it is not given, then it will default to strlen(*`string`*); i.e. end the replacing at the end of *`string`*. Of course, if *`length`* is zero then this function will have the effect of inserting *`replacement`* into *`string`* at the given *`start`* offset.

Return Values

The result string is returned. If *`string`* is an array then array is returned.

Examples

Example #83 - substr_replace() example

```
<?php
$var = 'ABCDEFGH:/MNRPQR/';
echo "Original: $var<hr />\n";

/* These two examples replace all of $var with 'bob'. */
echo substr_replace($var, 'bob', 0) . "<br />\n";
echo substr_replace($var, 'bob', 0, strlen($var)) . "<br />\n";
```

```
/* Insert 'bob' right at the beginning of $var. */
echo substr_replace($var, 'bob', 0, 0) . "<br />\n";

/* These next two replace 'MNRPQR' in $var with 'bob'. */
echo substr_replace($var, 'bob', 10, -1) . "<br />\n";
echo substr_replace($var, 'bob', -7, -1) . "<br />\n";

/* Delete 'MNRPQR' from $var. */
echo substr_replace($var, '', 10, -1) . "<br />\n";
?>
```

Notes

Note
This function is binary-safe.

See Also

- str_replace()
- substr()

substr

substr -- Return part of a string

Description

string **substr** (string `$string`, int `$start` [, int `$length`])

Returns the portion of *`string`* specified by the *`start`* and *`length`* parameters.

Parameters

`string`
The input string.

`start`
If *`start`* is non-negative, the returned string will start at the *`start`* 'th position in *`string`*, counting from zero. For instance, in the string ' *abcdef* ', the character at position *0* is ' *a* ', the character at position *2* is ' *c* ', and so forth. If *`start`* is negative, the returned string will start at the *`start`* 'th character from the end of *`string`*.

Example #84 - Using a negative *`start`*

```
<?php
$rest = substr("abcdef", -1);    // returns "f"
$rest = substr("abcdef", -2);    // returns "ef"
$rest = substr("abcdef", -3, 1); // returns "d"
?>
```

`length`
If *`length`* is given and is positive, the string returned will contain at most *`length`* characters beginning from *`start`* (depending on the length of *`string`*). If *`string`* is less than or equal to *`start`* characters long, **FALSE** will be returned. If *`length`* is given and is negative, then that many characters will be omitted from the end of *`string`* (after the start position has been calculated when a *`start`* is negative). If *`start`* denotes a position beyond this truncation, an empty string will be returned.

Example #85 - Using a negative *`length`*

```
<?php
$rest = substr("abcdef", 0, -1);  // returns "abcde"
$rest = substr("abcdef", 2, -1);  // returns "cde"
$rest = substr("abcdef", 4, -4);  // returns ""
$rest = substr("abcdef", -3, -1); // returns "de"
?>
```

Return Values

Returns the extracted part of string.

Examples

Example #86 - Basic substr() usage

```
<?php
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
echo substr('abcdef', 0, 8);  // abcdef
echo substr('abcdef', -1, 1); // f

// Accessing single characters in a string
// can also be achieved using "square brackets"
$string = 'abcdef';
echo $string[0];                 // a
echo $string[3];                 // d
echo $string[strlen($string)-1]; // f

?>
```

See Also

- strrchr()
- substr_replace()
- preg_match()
- trim()
- mb_substr()
- wordwrap()

trim

trim -- Strip whitespace (or other characters) from the beginning and end of a string

Description

string **trim** (string `$str` [, string `$charlist`])

This function returns a string with whitespace stripped from the beginning and end of *`str`*. Without the second parameter, trim() will strip these characters:

- " " (ASCII *32* (*0x20*)), an ordinary space.
- "\t" (ASCII *9* (*0x09*)), a tab.
- "\n" (ASCII *10* (*0x0A*)), a new line (line feed).
- "\r" (ASCII *13* (*0x0D*)), a carriage return.
- "\0" (ASCII *0* (*0x00*)), the *NUL* -byte.
- "\x0B" (ASCII *11* (*0x0B*)), a vertical tab.

Parameters

`str`
: The string that will be trimmed.

`charlist`
: Optionally, the stripped characters can also be specified using the *`charlist`* parameter. Simply list all characters that you want to be stripped. With*..* you can specify a range of characters.

Return Values

The trimmed string.

ChangeLog

Version	Description
4.1.0	The optional *`charlist`* parameter was added.

Examples

Example #87 - Usage example of trim()

```
<?php

$text   = "\t\tThese are a few words :) ...  ";
$binary = "\x09Example string\x0A";
$hello  = "Hello World";
var_dump($text, $binary, $hello);

print "\n";

$trimmed = trim($text);
var_dump($trimmed);

$trimmed = trim($text, " \t.");
var_dump($trimmed);

$trimmed = trim($hello, "Hdle");
var_dump($trimmed);

// trim the ASCII control characters at the beginning and end of $binary
// (from 0 to 31 inclusive)
$clean = trim($binary, "\x00..\x1F");
var_dump($clean);

?>
```

The above example will output:

```
string(32) "        These are a few words :) ...  "
string(16) "    Example string
"
string(11) "Hello World"

string(28) "These are a few words :) ..."
string(24) "These are a few words :)"
string(5) "o Wor"
string(14) "Example string"
```

Example #88 - Trimming array values with trim()

```
<?php
function trim_value(&$value)
{
   $value = trim($value);
}

$fruit = array('apple','banana ', ' cranberry ');
var_dump($fruit);
```

```
array_walk($fruit, 'trim_value');
var_dump($fruit);

?>
```

The above example will output:

```
array(3) {
 [0]=>
 string(5) "apple"
 [1]=>
 string(7) "banana "
 [2]=>
 string(11) " cranberry "
}
array(3) {
 [0]=>
 string(5) "apple"
 [1]=>
 string(6) "banana"
 [2]=>
 string(9) "cranberry"
}
```

See Also

- ltrim()
- rtrim()

ucfirst

ucfirst -- Make a string's first character uppercase

Description

string **ucfirst** (string `$str`)

Returns a string with the first character of *`str`* capitalized, if that character is alphabetic.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

Parameters

`str`
The input string.

Return Values

Returns the resulting string.

Examples

Example #89 - ucfirst() example

```
<?php
$foo = 'hello world!';
$foo = ucfirst($foo);             // Hello world!

$bar = 'HELLO WORLD!';
$bar = ucfirst($bar);             // HELLO WORLD!
$bar = ucfirst(strtolower($bar)); // Hello world!
?>
```

See Also

- lcfirst()
- strtolower()
- strtoupper()
- ucwords()

ucwords

ucwords -- Uppercase the first character of each word in a string

Description

string **ucwords** (string `$str`)

Returns a string with the first character of each word in *`str`* capitalized, if that character is alphabetic.

The definition of a word is any string of characters that is immediately after a whitespace (These are: space, form-feed, newline, carriage return, horizontal tab, and vertical tab).

Parameters

`str`
The input string.

Return Values

Returns the modified string.

Examples

Example #90 - ucwords() example

```
<?php
$foo = 'hello world!';
$foo = ucwords($foo);             // Hello World!

$bar = 'HELLO WORLD!';
$bar = ucwords($bar);             // HELLO WORLD!
$bar = ucwords(strtolower($bar)); // Hello World!
?>
```

Notes

Note

This function is binary-safe.

See Also

- strtoupper()
- strtolower()
- ucfirst()
- mb_convert_case()

vfprintf

vfprintf -- Write a formatted string to a stream

Description

int **vfprintf** (resource `$handle`, string `$format`, array `$args`)

Write a string produced according to *`format`* to the stream resource specified by *`handle`*.

Operates as fprintf() but accepts an array of arguments, rather than a variable number of arguments.

Parameters

`handle`

`format`
See sprintf() for a description of *`format`*.

`args`

Return Values

Returns the length of the outputted string.

Examples

Example #91 - vfprintf(): zero-padded integers

```
<?php
if (!($fp = fopen('date.txt', 'w')))
   return;

vfprintf($fp, "%04d-%02d-%02d", array($year, $month, $day));
// will write the formatted ISO date to date.txt
?>
```

See Also

- printf()
- sprintf()
- sscanf()
- fscanf()
- vsprintf()
- number_format()

vprintf

vprintf -- Output a formatted string

Description

int **vprintf** (string `$format`, array `$args`)

Display array values as a formatted string according to *`format`* (which is described in the documentation for sprintf()).

Operates as printf() but accepts an array of arguments, rather than a variable number of arguments.

Parameters

`format`
: See sprintf() for a description of *`format`*.

`args`

Return Values

Returns the length of the outputted string.

See Also

- printf()
- sprintf()
- vsprintf()

vsprintf

vsprintf -- Return a formatted string

Description

string **vsprintf** (string `$format`, array `$args`)

Operates as sprintf() but accepts an array of arguments, rather than a variable number of arguments.

Parameters

`format`
See sprintf() for a description of *`format`*.

`args`

Return Values

Return array values as a formatted string according to *`format`* (which is described in the documentation for sprintf()).

See Also

- sprintf()
- vprintf()

wordwrap

wordwrap -- Wraps a string to a given number of characters

Description

string **wordwrap** (string `$str` [, int `$width` [, string `$break` [, bool `$cut`]]])

Wraps a string to a given number of characters using a string break character.

Parameters

`str`
The input string.

`width`
The column width. Defaults to 75.

`break`
The line is broken using the optional *`break`* parameter. Defaults to ' *n* '.

`cut`
If the *`cut`* is set to **TRUE**, the string is always wrapped at the specified width. So if you have a word that is larger than the given width, it is broken apart. (See second example).

Return Values

Returns the given string wrapped at the specified column.

ChangeLog

Version	Description
4.0.3	The optional *`cut`* parameter was added.

Examples

Example #92 - wordwrap() example

```
<?php
$text = "The quick brown fox jumped over the lazy dog.";
$newtext = wordwrap($text, 20, "<br />\n");

echo $newtext;
?>
```

The above example will output:

```
The quick brown fox<br />
jumped over the lazy<br />
dog.
```

Example #93 - wordwrap() example

```
<?php
$text = "A very long woooooooooooord.";
$newtext = wordwrap($text, 8, "\n", true);

echo "$newtext\n";
?>
```

The above example will output:

```
A very
long
wooooooo
ooooord.
```

See Also

- nl2br()
- chunk_split()