Regular Expression (POSIX Extended)

Introduction

Tip
PHP also supports regular expressions using a Perl-compatible syntax using the PCRE functions. Those functions support non-greedy matching, assertions, conditional subpatterns, and a number of other features not supported by the POSIX-extended regular expression syntax.

Warning
These regular expression functions are not binary-safe. The PCRE functions are.

Regular expressions are used for complex string manipulation. PHP uses the POSIX extended regular expressions as defined by POSIX 1003.2. For a full description of POSIX regular expressions see the » regex man pages included in the regex directory in the PHP distribution. It's in manpage format, so you'll want to do something along the lines of *man /usr/local/src/regex/regex.7* in order to read it.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

Warning
Do not change the TYPE unless you know what you are doing.

To enable regexp support configure PHP *--with-regex[=TYPE]*. TYPE can be one of system, apache, php. The default is to use php.

The Windows version of PHP has built-in support for this extension. You do not need to load any additional extensions in order to use these functions.

Runtime Configuration

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

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Examples

Example #1 - Regular Expression Examples

```
<?php
// Returns true if "abc" is found anywhere in $string.
ereg("abc", $string);

// Returns true if "abc" is found at the beginning of $string.
ereg("^abc", $string);

// Returns true if "abc" is found at the end of $string.
ereg("abc$", $string);

// Returns true if client browser is Netscape 2, 3 or MSIE 3.
eregi("(ozilla.[23]|MSIE.3)", $_SERVER["HTTP_USER_AGENT"]);

// Places three space separated words into $regs[1], $regs[2] and $regs[3].
ereg("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)", $string, $regs);

// Put a <br /> tag at the beginning of $string.
$string = ereg_replace("^", "<br />", $string);

// Put a <br /> tag at the end of $string.
$string = ereg_replace("$", "<br />", $string);

// Get rid of any newline characters in $string.
$string = ereg_replace("\n", "", $string);
?>
```

POSIX Regex Functions

See Also

For regular expressions in Perl-compatible syntax have a look at the PCRE functions. The simpler shell style wildcard pattern matching is provided by fnmatch().

ereg_replace

ereg_replace -- Replace regular expression

Description

string **ereg_replace** (string `$pattern`, string `$replacement`, string `$string`)

This function scans *`string`* for matches to *`pattern`*, then replaces the matched text with *`replacement`*.

Parameters

`pattern`
: A POSIX extended regular expression.

`replacement`
: If *`pattern`* contains parenthesized substrings, *`replacement`* may contain substrings of the form \\ *digit*, which will be replaced by the text matching the digit'th parenthesized substring; *0* will produce the entire contents of string. Up to nine substrings may be used. Parentheses may be nested, in which case they are counted by the opening parenthesis.

`string`
: The input string.

Return Values

The modified string is returned. If no matches are found in *`string`*, then it will be returned unchanged.

Examples

For example, the following code snippet prints "This was a test" three times:

Example #2 - ereg_replace() example

```
<?php

$string = "This is a test";
echo str_replace(" is", " was", $string);
echo ereg_replace("( )is", "\\1was", $string);
echo ereg_replace("(( )is)", "\\2was", $string);

?>
```

One thing to take note of is that if you use an integer value as the *replacement* parameter, you may not get the results you expect. This is because ereg_replace() will interpret the number as the ordinal value of a character, and apply that. For instance:

Example #3 - ereg_replace() example

```
<?php
/* This will not work as expected. */
$num = 4;
$string = "This string has four words.";
$string = ereg_replace('four', $num, $string);
echo $string;   /* Output: 'This string has   words.' */

/* This will work. */
$num = '4';
$string = "This string has four words.";
$string = ereg_replace('four', $num, $string);
echo $string;   /* Output: 'This string has 4 words.' */
?>
```

Example #4 - Replace URLs with links

```
<?php
$text = ereg_replace("[[:alpha:]]+://[^<>[:space:]]+[[:alnum:]/]",
                     "<a href=\"\\0\">\\0</a>", $text);
?>
```

Notes

Tip

preg_replace(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to ereg_replace().

See Also

- ereg()
- eregi()
- eregi_replace()
- str_replace()
- preg_match()

ereg

ereg -- Regular expression match

Description

int **ereg** (string `$pattern`, string `$string` [, array `&$regs`])

Searches a *`string`* for matches to the regular expression given in *`pattern`* in a case-sensitive way.

Parameters

`pattern`
: Case sensitive regular expression.

`string`
: The input string.

`regs`
: If matches are found for parenthesized substrings of *`pattern`* and the function is called with the third argument *`regs`*, the matches will be stored in the elements of the array *`regs`*. $regs[1] will contain the substring which starts at the first left parenthesis; $regs[2] will contain the substring starting at the second, and so on. $regs[0] will contain a copy of the complete string matched.

Return Values

Returns the length of the matched string if a match for *`pattern`* was found in *`string`*, or **FALSE** if no matches were found or an error occurred.

If the optional parameter *`regs`* was not passed or the length of the matched string is 0, this function returns 1.

Examples

Example #5 - ereg() example

The following code snippet takes a date in ISO format (YYYY-MM-DD) and prints it in DD.MM.YYYY format:

```
<?php
if (ereg ("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
   echo "$regs[3].$regs[2].$regs[1]";
} else {
```

```
    echo "Invalid date format: $date";
}
?>
```

Notes

Note
preg_match(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to ereg().

Note
Up to (and including) PHP 4.1.0 *$regs* will be filled with exactly ten elements, even though more or fewer than ten parenthesized substrings may actually have matched. This has no effect on ereg() 's ability to match more substrings. If no matches are found, *$regs* will not be altered by ereg().

See Also

- eregi()
- ereg_replace()
- eregi_replace()
- preg_match()
- strpos()
- strstr()
- quotemeta()

eregi_replace

eregi_replace -- Replace regular expression case insensitive

Description

string **eregi_replace** (string `$pattern`, string `$replacement`, string `$string`)

This function is identical to ereg_replace() except that this ignores case distinction when matching alphabetic characters.

Parameters

`pattern`
A POSIX extended regular expression.

`replacement`
If *`pattern`* contains parenthesized substrings, *`replacement`* may contain substrings of the form \\ *digit*, which will be replaced by the text matching the digit'th parenthesized substring; *0* will produce the entire contents of string. Up to nine substrings may be used. Parentheses may be nested, in which case they are counted by the opening parenthesis.

`string`
The input string.

Return Values

The modified string is returned. If no matches are found in *`string`*, then it will be returned unchanged.

Examples

Example #6 - Highlight search results

```
<?php
$pattern = '(>[^<]*)('. quotemeta($_GET['search']) .')';
$replacement = '\\1<span class="search">\\2</span>';
$body = eregi_replace($pattern, $replacement, $body);
?>
```

See Also

- ereg()
- eregi()
- ereg_replace()

eregi

eregi -- Case insensitive regular expression match

Description

int **eregi** (string `$pattern`, string `$string` [, array `&$regs`])

This function is identical to ereg() except that it ignores case distinction when matching alphabetic characters.

Parameters

`pattern`
Case insensitive regular expression.

`string`
The input string.

`regs`
If matches are found for parenthesized substrings of *`pattern`* and the function is called with the third argument *`regs`*, the matches will be stored in the elements of the array *`regs`*. $regs[1] will contain the substring which starts at the first left parenthesis; $regs[2] will contain the substring starting at the second, and so on. $regs[0] will contain a copy of the complete string matched.

Return Values

Returns the length of the matched string if a match for *`pattern`* was found in *`string`*, or **FALSE** if no matches were found or an error occurred.

If the optional parameter *`regs`* was not passed or the length of the matched string is 0, this function returns 1.

Examples

Example #7 - eregi() example

```
<?php
$string = 'XYZ';
if (eregi('z', $string)) {
   echo "'$string' contains a 'z' or 'Z'!";
}
?>
```

See Also

- ereg()
- ereg_replace()
- eregi_replace()
- stripos()
- stristr()

split

split -- Split string into array by regular expression

Description

array **split** (string `$pattern`, string `$string` [, int `$limit`])

Splits a *`string`* into array by regular expression.

Parameters

`pattern`
: Case sensitive regular expression. If you want to split on any of the characters which are considered special by regular expressions, you'll need to escape them first. If you think split() (or any other regex function, for that matter) is doing something weird, please read the file *regex.7*, included in the *regex/* subdirectory of the PHP distribution. It's in manpage format, so you'll want to do something along the lines of *man /usr/local/src/regex/regex.7* in order to read it.

`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 whole rest of *`string`*.

Return Values

Returns an array of strings, each of which is a substring of *`string`* formed by splitting it on boundaries formed by the case-sensitive regular expression *`pattern`*.

If there are *n* occurrences of *`pattern`*, the returned array will contain *n +1* items. For example, if there is no occurrence of *`pattern`*, an array with only one element will be returned. Of course, this is also true if *`string`* is empty. If an error occurs, split() returns **FALSE**.

Examples

Example #8 - split() example

To split off the first four fields from a line from */etc/passwd*:

```
<?php
list($user, $pass, $uid, $gid, $extra) =
```

```
   split(":", $passwd_line, 5);
?>
```

Example #9 - split() example

To parse a date which may be delimited with slashes, dots, or hyphens:

```
<?php
// Delimiters may be slash, dot, or hyphen
$date = "04/30/1973";
list($month, $day, $year) = split('[/.-]', $date);
echo "Month: $month; Day: $day; Year: $year<br />\n";
?>
```

Notes

Tip

preg_split(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to split(). If you don't require the power of regular expressions, it is faster to use explode(), which doesn't incur the overhead of the regular expression engine.

Tip

For users looking for a way to emulate Perl's *@chars = split('', $str)* behaviour, please see the examples for preg_split() or str_split().

See Also

- preg_split()
- spliti()
- str_split()
- explode()
- implode()
- chunk_split()
- wordwrap()

spliti

spliti -- Split string into array by regular expression case insensitive

Description

array **spliti** (string `$pattern`, string `$string` [, int `$limit`])

Splits a *`string`* into array by regular expression.

This function is identical to split() except that this ignores case distinction when matching alphabetic characters.

Parameters

`pattern`
: Case insensitive regular expression. If you want to split on any of the characters which are considered special by regular expressions, you'll need to escape them first. If you think spliti() (or any other regex function, for that matter) is doing something weird, please read the file *regex.7*, included in the *regex/* subdirectory of the PHP distribution. It's in manpage format, so you'll want to do something along the lines of *man /usr/local/src/regex/regex.7* in order to read it.

`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 whole rest of *`string`*.

Return Values

Returns an array of strings, each of which is a substring of *`string`* formed by splitting it on boundaries formed by the case insensitive regular expression *`pattern`*.

If there are *n* occurrences of *`pattern`*, the returned array will contain *n +1* items. For example, if there is no occurrence of *`pattern`*, an array with only one element will be returned. Of course, this is also true if *`string`* is empty. If an error occurs, spliti() returns **FALSE**.

Examples

This example splits a string using 'a' as the separator :

Example #10 - spliti() example

```
<?php
$string = "aBBBaCCCADDDaEEEaGGGA";
$chunks = spliti ("a", $string, 5);
print_r($chunks);
?>
```

The above example will output:

```
Array
(
 [0] =>
 [1] => BBB
 [2] => CCC
 [3] => DDD
 [4] => EEEaGGGA
)
```

See Also

- preg_split()
- split()
- explode()
- implode()

sql_regcase

sql_regcase -- Make regular expression for case insensitive match

Description

string **sql_regcase** (string `$string`)

Creates a regular expression for a case insensitive match.

Parameters

`string`
The input string.

Return Values

Returns a valid regular expression which will match *`string`*, ignoring case. This expression is *`string`* with each alphabetic character converted to a bracket expression; this bracket expression contains that character's uppercase and lowercase form. Other characters remain unchanged.

Examples

Example #11 - sql_regcase() example

```
<?php
echo sql_regcase("Foo - bar.");
?>
```

The above example will output:

```
[Ff][Oo][Oo] - [Bb][Aa][Rr].
```

This can be used to achieve case insensitive pattern matching in products which support only case sensitive regular expressions.