2022-01-16

NLNLTD

PHP Programming

A Step-by-Step Guide to Learn, in an Easy Way, the Fundamentals of PHP Programming Language

3ND Edition

Daniel Robinson

PHP Programming

A Step-by-Step Guide to Learn, in an Easy Way, the Fundamentals of PHP Programming Language

3nd Edition

By

Daniel Robinson

Content:

Part 1. Language syntax and operators

Expressions

if
Allows you to organize the execution of code snippets conditionally.

Syntax:
if (*expression*) statement

Can be nested indefinitely in other IFs.

```
if ($ a> $ b)
  print "$ a is greater than $ b";
if ($ a> $ b) {echo "$ a is greater than $ b; $ b = $ a;}
```

else
Expands the IF capabilities in terms of handling variants of an expression when it is FALSE.
The ELSE expression is executed only if the IF is FALSE.

```
if ($ a > $ b ) {
  echo "a is greater than b ";
} else {
  echo "a is less than b";
}
```

elseif
Is a combination of IF and ELSE. Allows the expression to be executed if the IF value is FALSE, but unlike ELSE it is executed if the ELSEIF expression is TRUE.

```
if ($ a> $ b) {
  echo "a is greater than b";
} elseif ($ a == $ b) {
  echo "a is equal to b";
} else {
  echo "a is less than b";
}
```

if_endif
One of the possible options for grouping statements with an IF statement.

Convenient for embedding large blocks of HTML code inside an IF statement.

```
if ($ a == 1):
  echo "a is 1";
elseif ($ a == 2):
echo "a is 2";
else:
  echo "not equal to 1 and 2";
endif;
<? php if ($ a == 5):?> A = 5 <? php endif;?>
-Block HTML code A = 5 will be visible,
    if the condition $ a == 5 is true
```

while
The simplest type of loop in PHP. Forces PHP to execute nested statements as long as the *condition* is TRUE. If the *condition* is FALSE from the very beginning, then the loop will not be executed once.

Syntax: WHILE (*condition*) expression

You can group multiple statements within curly braces, or use **an alternative syntax:**
WHILE (*condition*) expression ... ENDWHILE;

```
$ a = 1;
while ($ a <= 5) {
  echo $ a ++; }
$ a = 1;
while ($ a <= 5):
  echo $ a;
  $ a ++;
endwhile;
```
- These two examples print numbers from 1 to 5.

do_while
A loop similar to WHILE, but the value of the logical expression is checked not before, but after the end of the iteration. The main difference is that the loop will be executed at least once.

```
$ a = 1;
do {
  echo $ a;
} while ($ a> 1);
```
You can stop using the statement block in the middle by embedding the **BREAK** statement in the **DO..WHILE (0)** loop :
```
do {
  if ($ a == 5) {
    ech o "A equals 5"
    break;
  }
  $ a * = $ b;
  if ($ a <$ minimum) {
    break;
  }
  echo "A is $ a";
} while (0);
```

for
The most powerful loop in PHP.

Syntax:
FOR (*condition1* ; *condition2* ; *condition3*) of expression

condition1 - Unconditionally executed (evaluated) at the beginning of the loop
condition2 - Checked at the beginning of each iteration. If it is TRUE, then the loop continues and the nested statements are executed. If it is FALSE, then the loop ends.
condition3 - Executed (evaluated) at the end of each iteration.
Each of these conditions can be empty.
Example 1:
```
for ($ a = 1; $ a <= 5; $ a ++) {
  echo $ a;
}
```

Example 2:
```
for ($ a = 1 ;; $ a ++) {
  if ($ a> 5) {
    break;
  }
  echo $ a;
}
```
Example 3:
```
$ a = 1;
for (;;) {
```

```
  if ($ a> 5) {
    break;
  }
  print $ a;
  $ a ++;
}
```

Example 4:

```
for ($ a = 1; $ a <= 5; print $ a, $ a ++);
```

PHP supports an alternative syntax FOR:

FOR (*conv1; conv2; conv3;*): operators; ...; ENDFOR;

break Break the
execution of the current loop.

Example :

```
$ a = 0;
while ($ a <5) {
  if ($ arr [$ a] == "stop") {
    break;
  }
  $ a ++;
}
```

continue
Goes to the beginning of the next cycle.

```
while (list ($ key, $ value) = each ($ arr)) {
  if ($ key% 2) {
    continue;
  }
  do_something_odd ($ value);
}
```

switch
Compares a variable or expression with different values and executes different pieces of code depending on what the expression's value equals to.

```
switch ($ a) {
  case 0:
    echo "A is 0";
    break;
  case 1:
    echo "A is 1";
    break;
  case 2:
    echo "A is 2";
    break;
  default:
    echo "A is not equal to 0, 1, 2";
}
```

default - matches all values that do not satisfy other CASEs. CASE - can be of any scalar type, i.e. integers or floating point numbers and strings.

require
Replaces itself with the contents of the specified file.

Example:

```
require ("include.inc");
```

But you can't put it inside a loop and expect it to include the contents of another file multiple times during each iteration. There is INCLUDE for this.

include
Inserts and executes the contents of the specified file.

```
$ files = array ("first.inc", "seco nd.inc", "third.inc");
for ($ a = 0; $ a <count ($ files); $ a ++) {
  include ($ files [$ a ]);
}
```

Because INCLUDE () is a special operator, you must enclose it in curly braces when used inside a conditional operator.

```
if ($ a <5) {
  include ("file_1.in c");
} else {
  include ("file_2.inc");
}
```

function
A **function** declaration.

Any valid PHP code can be inside a function, even a declaration of another function or class. Functions must be declared before they can be referenced.

```
function foo ($ arg_1, $ arg_2, ..., $ arg_n) {
  echo " Sample function .";
  return $ retvalue;
}
```

Returning Results:
Results are returned via an optional return statement.
The returned result can be of any type, including lists and objects.

```
function my_sqrt ($ num) {
  retu rn $ num * $ num;
}
echo my _ sqrt (4); // will print 16
```

Multiple results cannot be returned as a result, but you can implement this by returning a list:

```
function foo () {
  return array (0, 1, 2);
}
list ($ zero, $ one, $ two) = foo ();
```

Arguments:
Information can be passed to a function through an argument list, which is a comma-separated list of variables and / or constants.
Variable length argument lists are not supported, but the same can be achieved by passing arrays.

```
function takes_array ($ input) {
  echo "$ input [0] + $ input [1] =", $ input [0] + $ input [1];
}
```

Pass by reference:
By default, function arguments are passed by value. To change the arguments in a function, they must be passed by reference.
To do this, put an ampersand (&) in front of the argument name in the function declaration:

```
function foo (& $ bar ) {
  $ bar. = "and an extra string.";
}
$ str = "This is a string,";
foo ($ str);
echo $ str; // will output: "This is a string, and an extra string."
function foo ($ bar ) {
```

```
  $ bar. = "and an extra string.";
}
$ str = "This is a string,";
foo ($ str);
echo $ str; // will output: "This is a string,"
foo (& $ str);
echo $ str; // prints: "This is a string, and an extra string."
```

Default values: The ***default***
value must be a constant, not a variable or a member of a class.

```
function day ($ type = " monday ") {
  echo "Today is $ type .";
}
echo day (); // will output: Today is Monday.
echo day ("Tuesday"); // will output: Today is Tuesday.
```

The default arguments in the description must appear to the right of the other arguments.

```
function day ($ day_num, $ type = " Monday ") {
  return " Today $ day_num is $ type.";
}
```

old_function
The OLD_FUNCTION statement allows you to define a function using PHP / FI2 syntax (except that you must replace "function" with "old_function").

This property is for compatibility only and should only be used by PHP / FI2 -> PHP3 converters. Functions described in this way cannot be called from the PHP service code. You can get around this by introducing a special function in PHP3 terms that will call OLD_FUNCTION.

class
A set of variables and functions that operate on these variables.

```
<? php
class Cart {
  var $ items; // The number of items in the shopping cart
  // Add $ num items of type $ artnr to cart
  function add_item ($ artnr, $ num) {
    $ this-> items [$ artnr] + = $ num;
  }
  // Remove $ num $ artnr items from the cart
  function remove_item ($ artnr, $ num) {
    if ($ this-> items [$ artnr]> $ num) {
        $ this-> items [$ artnr] - = $ num;
        return true;
    } else {
     return false;
    }
  }
}
?>
```

Classes are types, that is, stubs for real variables. You must create variables of the desired type using the new operator:
$ cart = new Cart;
$ cart-> add_item ("10", 1);

Classes can be extensions of other classes. An extended class has all the variables and functions of the base class and what you define when extending the class. This is done using the extends keyword:

```
class Named _ Cart extends Cart {
  var $ owner;
  function set_owner ($ name) {
    $ t his -> owner = $ name ;
  }
```

```
}
```

This defines the Named_Cart class, which has all the variables and functions of the Cart class plus an additional $ owner variable and an additional set_owner () function. You can create a named cart in the usual way and set or get the owner of the cart. You can also use the normal cart functions in a named cart:

```
$ ncart = new Named_Cart; // Create cart
$ ncart-> set_owner ( "kris" ); // Specify the owner of the print
$ ncart-> owner; // Print the name of the cart owner
$ ncart-> add_item ("10", 1); // inherited from regular cart
```

Part 2. Functions for working with data

Math functions

Rounding functions

abs
Returns the modulus of a number.

Syntax:
mixed abs (mixed $ number) The

type of the *$ number* parameter can be float or int, and the return type is always the same as the type of this parameter.

```
$ x = abs (-4); // $ x = 4
$ x = abs (-7.45); // $ x = 7.45
```

round
Rounds a fraction to an integer.

Syntax:
double round (double $ val)

Rounds *$ val* to the nearest integer and returns the result.

```
$ foo = round (3.4); // $ foo == 3.0
$ foo = round (3.5); // $ foo == 4.0
$ foo = round (3.6); // $ foo == 4.0
$ x = round (5.3); // $ x = 5
$ x = round (5.4); // $ x = 5
$ x = round (5.45); // $ x = 5
$ x = round (5.5); // $ x = 6
```

ceil
Completion of a fraction to the next integer.

Syntax:
int ceil (float $ number)

Returns the smallest integer, at least *$ number* . Of course, passing an integer to *$ number is* pointless.

```
$ x = ceil (5.0); // $ x = 5
$ x = ceil (5.1); // $ x = 6
$ x = ceil (5.9); // $ x = 6
```

floor
Remove the fractional part of a number.

Syntax:
int floor (float $ number)

Returns the maximum integer not exceeding *$ number* .

```
$ x = floor (5.1); // $ x = 5
$ x = floor (5.9); // $ x = 5
```

Random numbers

srand
Initializes the random number generator.

Syntax:
void srand (int seed)

Initializes the random number generator to seed.

```
srand ((double) microtime () * 1000000);
$ random = rand ();
echo $ random ;
```

Here's what you get:

```
5645
```

getrandmax
Returns the largest possible random number.

Syntax:
int getrandmax ()

This function returns the maximum value that can be obtained using the random number generating function rand ().
Usually it is 32767

rand
Generates a random number.

Syntax:
int rand ([int max [, int min]])

When called with the optional min and max parameters, this function generates a random number within those parameters, inclusive.
If the min and max parameters are not present, a number between 0 and RAND_MAX is returned.
For this function to work correctly, before using it, you need to initialize the random number generator with the srand () function.

mt_rand
The function returns an MT-random number, even enough even to be used in cryptography.

Syntax:
int mt_rand (int $ min = 0, int $ max = RAND_MAX)

If you want to generate numbers not from 0 to *RAND_MAX* (this constant specifies the maximum allowed random number, and can be obtained by calling **mt_getrandmax ()**), set the appropriate interval in parameters *$ min* and *$ max* . Don't forget to just run **mt_srand ()** before the first call to this function .

```
mt_srand (time () + (double) microtime () * 1000000);
$ x = mt_rand (1,100); // $ x is a value from 1 to 100
```

mt_srand
Sets the MT random number generator to a new sequence.

Syntax:
void mt_srand (int seed)

The fact is that although the numbers generated by mt_rand () are fairly equally probable, they have one drawback: the sequence of generated numbers will be the same if the script is called several times in a row. The **mt_srand ()** function solves this problem: it selects a new sequence based on the *$ seed* parameter , and in an almost unpredictable way.

```
mt_srand (time () + (double) microtime () * 1000000);
for ($ i = 0; $ i <= 10; $ i ++) {
  $ x = mt _rand (1,10);
};
```

In this case, the sequence is set based on the script startup time (in seconds), so it is rather unpredictable. For an even more reliable result, it is recommended to add more microseconds here (which was done), as well as the identifier of the process that called the script.

mt_getrandmax
Returns the maximum MT random number.

Syntax:
int mt_getrandmax ()

Returns the maximum number that can be generated by the mt_rand () function - in other words, the *RAND_MAX* constant

```
$ max = mt_getrandmax ();
// $ max = 2147483647
```

lcg_value
function generates a random fractional number.

Syntax:
double lcg_value ()

This function returns a pseudo-random fractional number in the range 0 to 1.

Translation into various number systems

base_convert
Converting a number from one number system to another.

Syntax:
string base_convert (string $ number, int $ frombase, int $ tobase)

Converts *$ number* (specified as a base string in base *$ frombase*) to base *$ tobase* . The *$ frombase* and *$ tobase parameters* can only take values between 2 and 36, inclusive. In the string *$ number, the* numbers stand for themselves, and the letter *a* matches 11, *b* -12, etc. to *z* , which stands for 36. For example, the following commands will print 11111111 (8 ones), because this is nothing more than the binary representation of the hexadecimal number *FF* :

```
$ x = base_convert ("FF", 16.2); // $ x = 11111111
$ x = base_convert ("11111111", 2.16); // $ x = FF
$ x = base_convert ("200", 10 , 16); // $ x = C8
```

bindec Converts
binary to decimal.

Syntax:
int bindec (string binary_string)

Converts the binary number specified in *binary_string* to a decimal number. The maximum number that can still be converted is 2 147 483 647

```
$ x = bindec (11111111); // $ x = 255
$ x = bindec (10101010); // $ x = 170
$ x = bindec (2147483647); // $ x = 11111111111111111111111111111111
```

decbin Convert
decimal to binary.

Syntax:
string decbin (int $ number)

Returns a string representing the binary representation of the integer *$ number* . The maximum number that can still be converted is 2147483647, which looks like 31 ones in binary.

There are similar functions for octal and hexadecimal systems. They are called the same, only instead of

"bin" are substituted respectively "oct" and "hex".

```
$ x = decbin (255); // $ x = 11111111
$ x = decbin (2147483647); // $ x = 11111111111111111111111111111111
```

dechex
Performs decimal to hexadecimal conversion.

Syntax:
string dechex (int number)

Returns a string representing the hexadecimal representation of the integer *number* . The maximum number that can still be converted is 2,147,483,647

```
$ x = dechex (2147483647); // $ x = 7 fffffff
```

decoct
Performs decimal to octal conversion.

Syntax:
string decoct (int number)

Returns a string representing the octal representation of the integer *number* . The maximum number that can still be converted is 2,147,483,647

```
$ x = dechex (2147483647); // $ x = 17777777777
```

hexdec Convert a
hexadecimal number to decimal.

Syntax:
int hexdec (string hex_string)

Converts the hexadecimal number specified in *hex_string* to a decimal number. The maximum number that can still be converted is 7fffffff

```
$ x = hexdec (7 fffffff ); // $ x = 2147483647
```

octdec Convert
an octal number to decimal.

Syntax:
int octdec (string octal_string)

Converts the octal number specified in *octal_string* to a decimal number. The maximum number that can still be converted is 17777777777

```
$ x = octdec (17777777777); // $ x = 2147483647
```

deg2rad Converts
degrees to radians.

Syntax:
double deg2rad (double number)

Converts the degrees specified in the *number* parameter to radians.

rad2deg Converts
radians to degrees.

Syntax:
double rad2deg (double number)

Converts the radians specified in the *number* parameter to degrees.

number_format
Formatting a number.

Syntax:
number_format ($ number, $ decimals, $ dec_point = ".", $ Thousands_sep = ",");

This function formats a floating point number by dividing it into triads with the specified precision. It can be called with two or four arguments, but not three! The *$ decimals parameter* specifies how many decimal places the number should have in the output string. The *$ dec_point parameter* is the integer and fractional separator, and the *$ thousands_sep* parameter is the triad separator in the number (if you specify an empty string in its place, the triads are not separated from each other).

Minimum and maximum

min
This function returns the smallest of the numbers given in its arguments.

Syntax:
mixed min (mixed $ arg1 [int $ arg2, ..., int $ argn])

There are two ways to call this function: with one parameter or with several parameters. If only one parameter (the first) is specified, then it must be an array and the minimum element of this array is returned. Otherwise, the first (and other) arguments are treated as floating point numbers, they are compared, and the smallest is returned. The return type is chosen as follows: if at least one of the numbers passed to the input is specified in floating point format, then the result will be floating point, otherwise the result will be an integer. You cannot lexographically compare strings with this function - only numbers.

```
$ x = min (5,3,4,6,5,6,8,9);
// $ x = 3
$ x [0] = 4;
$ x [1] = 1;
$ x [2] = 5;
$ x [3] = 2;
echo min ($ x); // will output 1
```

max
Get the largest argument.

Syntax:
mixed max (mixed $ arg1 [int $ arg2, ..., int $ argn]) This

function works like **min ()** , only it looks for the maximum value.

```
$ x = max (5,3,4,6,5,6,8,9);
// $ x = 9
$ x [0] = 4;
$ x [1] = 1;
$ x [2] = 5;
$ x [3] = 2;
echo max ($ x); // will print 5
```

Power functions

sqrt
Returns the square root of the argument.

Syntax:
float sqrt (float $ arg)

If the argument is negative, a warning is generated, but the program does not terminate!

```
$ x = sqrt (9); // $ x = 3
echo s qrt (25); // will print 5
```

```
echo sqrt (-25); // prints -1. # IND
```

log
Returns the natural logarithm of the argument.

Syntax:
float log (float $ arg)

Prints a warning if the number is invalid, but does not terminate the program.

```
$ x = log (exp (2)); // exp (2) - e to the power of 2
                     // $ x = 2
$ x = log (M_E); // $ x = 1
echo log (10); // will output 2.302585092994
```

log10
Returns the decimal logarithm of the argument.

Syntax:
float log10 (float $ arg)

Prints a warning if the number is invalid, but does not terminate the program.

```
echo log10 (100); // will output 2
```

exp
Returns e (2.718281828) to the power of *$ arg* .

Syntax :
float exp (float $ arg)

```
$ x = exp (1); // $ x = 2.718281828459
```

pow
Exponentiation.

Syntax:
float pow (float $ base, float $ exp)

Returns *$ base* to the power of *$ exp* .

```
$ x = pow (3,2); // $ x = 9
$ x = pow ("3", 2); // $ x = 9
```

Trigonometry

sin
Returns the sine of the argument.

Syntax:
float sin (float $ arg) The

argument is in radians.

```
$ x = sin (M_PI_2); // $ x = 1
```

cos
Returns the cosine of the argument.

Syntax:
float cos (float $ arg)

```
$ x = cos (0); // $ x = 0
$ x = cos (M_PI); // $ x = -1
```

tan

Returns the tangent of the argument, given in radians.

Syntax :
float tan (float $ arg)
```
$ x = tan (M_PI_4); // $ x = 1
```

acos
Returns the arc cosine of the argument.

Syntax:
float acos (float $ arg)
```
$ x = acos (0); // $ x = pi / 2
$ x = acos (1); // $ x = 0
```

asin
Returns the inverse sine.

Syntax:
float asin (float $ arg)
```
$ x = asin (0); // $ x = 0
$ x = asin (1); // $ x = pi / 2
```

atan
Returns the arctangent of the argument.

Syntax:
flo at atan (float $ arg)
```
$ x = atan (0); // $ x = 0
$ x = atan (1); // $ x = pi / 4
```

atan2
Get the arctangent of two numbers.

Syntax:
float atan2 (float $ y, float $ x)

Returns the arctangent of *$ y / $ x* , but including the quarter that point (*$ x, $ y*) is in. This function returns the result in radians belonging to the segment from -? before ?.
```
$ x = atan2 (1,1); // $ x = pi / 4
$ x = atan2 (-1, -1); // $ x = -3 * pi / 4
```

pi
Returns pi - 3.14.

Syntax:
double pi ()

This function must be called with a pair of empty parentheses:

```
$ x = pi () * 2 // $ x = 31.415926535898
```

Advanced Precision BCMath Functions

bcadd Adds
two arbitrary precision numbers.

Syntax:
string bcadd (string left_operand, string right_operand [, int scale]);

This function returns a string representation of the sum of the two parameters (left_operand + right_operand) with the precision specified in the optional parameter scale.
The scale indicates the number of decimal places).

bccomp
Comparison of two arbitrary precision numbers.

Syntax :
int bccomp (string left_operand, string right_operand, [int scale]);

Compares the number (left_operand with right_operand) and returns a result of type integer (an integer). The scale parameter is used to set the number of digits after the decimal point used in the comparison. If the two parts are equal, 0 is returned. If the left part is greater than the right part, +1 is returned, and if the left part is less than the right part, -1 is returned.

bcdiv
The division operation for two arbitrary precision numbers.

Syntax :
string bcdiv (string left_operand, string right_operand [, intscale]);

Divides left_operand by right_operand and returns the result with the precision (decimal places) specified in the scale parameter.

bcmod
Returns the remainder of an integer division.

Syntax:
string bcmod (left_operand, string modulus);

This function returns the remainder of the integer division of left_operand by modulus.

bcmul Multiplication
operation for two arbitrary precision numbers.

Syntax:
string bcmul (string left_operand, string right_operand [, int scale]);

Multiplies left_operand by right_operand, and outputs the result as a string with the precision specified in the scale variable.

bcpow Raises
one arbitrary precision number to the power of another.

Syntax:
string bcpow (string x, string y, [int scale]);

Raising x to the power of y. The scale parameter can be used to set the number of digits after the dot.

bcscale
Sets the precision of the calculation.

Syntax:
string bcscale (int scale);

This function sets the default precision for all BCMath math functions that do not explicitly define precision.

bcsqrt

Get the square root of an arbitrary precision number.

Syntax:
string bcsqrt (string operand [, int scale]);

Returns the square root of operand. The scale parameter sets the number of digits after the decimal point in the result.

bcsub
Subtracts one arbitrary precision number from another.

Syntax:
string bcsub (string left_operand, right_operand [, int scale]);

Returns the difference of the two variables specified in the parameters of the function (left_operand - right_operand) with the precision specified in the optional parameter scale.

GMP functions

Functions of this kind allow working with high precision integers of a certain format using the GNU MP library.
This library is not included in the standard PHP package. You can download library codes and documentation from http://www.swox.com/gmp/ .

The functions provided in this library can also work with regular integer arguments. In this case, they will be automatically converted to GMP format. However, to improve performance, it is still recommended to use GMP numbers.

GMP functions. GMP Function Values

gmp_init
Creates a GMP number.

Syntax:
resource gmp_init (mixed number)

The GMP number is generated from an integer or string argument.
The string can contain a number in decimal or hexadecimal format. If it is in hexadecimal format, the number must be prefixed with 0x.

```
$ x = gmp_init (45);
$ y = gmp_init ("46");
$ z = gmp _ init ("0 xfa 4 b ");
```

This function is optional (arguments are automatically converted to GMP format), but desirable (performance is improved by using gmp_init ()).

gmp_intval
Convert GMP number to integer.

Syntax:
int gmp_intval (resource gmpnumber)

This function converts a GMP number to an integer if the received number does not exceed its maximum size.

gm p_strval
Convert GMP number to string.

Syntax:

string gmp_strval (resource gmpnumber [, int base])

The function returns the gmpnumber number in string format specified in the optional base parameter.
Returns in decimal by default).
The base parameter can take values from 2 to 36.

```
$ x = gmp_init ("0xf1a5");
echo " In decimal :" .gmp_strval ($ x);
echo " In base-36:" .gmp_strval ($ x, 36);
```

gmp_abs
Calculates the modulus of a GMP number.

Syntax:
resource gmp_abs (resourse x)

Returns the absolute value of the number specified in the x parameter.

gmp_sign
Returns the sign of a number.

Syntax:
int gmp_sign (resource x)

gmp_sign () will return 1 if x is positive, and 0 if negative.

gmp_neg
Returns the negative value of a number.

Syntax :
recource gmp_neg (resource x)

Will return -x.

GMP functions. Arithmetic

gmp_add Adds
two numbers.

Syntax:
recource gmp_add (resource x, recource y)

The function will return a GMP number equal to the sum of the x and y arguments.

gm p_sub
Subtract two numbers.

Syntax:
recource gmp_sub (resource x, recource y)

The function will return a GMP number equal to the difference between the x and y arguments.

gmp_mul
Multiplication of two numbers.

Syntax:
recource gmp_mul (resource x, recource y)

The function will return a GMP number equal to the product of the x and y arguments.

gmp_div

Divide two numbers.

Syntax:
recource gmp_div (resource x, recource y [, int round])

The function will return a GMP number equal to the division of the x arguments by y. Depending on the optional round parameter, the division result will be rounded as follows:
- GMP_ROUND_ZERO - numbers after the dot are discarded
- GMP_ROUND_PLUSINF - the division result is rounded up
- GMP_ROUND_MINUSINF - division result is rounded down

This function is synonymous with gmp_div_q ().

gmp_div_q
Divide two numbers.

Syntax:
recource gmp_div_q (resource x, recource y [, int round])

The function will return a GMP number equal to the division of the x arguments by y. Depending on the optional round parameter, the division result will be rounded as follows:
- GMP_ROUND_ZERO - numbers after the dot are discarded
- GMP_ROUND_PLUSINF - the division result is rounded up
- GMP_ROUND_MINUSINF - the division result is rounded down

This function has a synonym - gmp_div ().

gmp_div_r
Return the remainder of an integer division.

Syntax:
recource gmp_div_r (resource x, recource y [, int round])

The function returns the remainder of dividing x by y. The sign will inherit from the x argument.

gmp_div_q r Divide
with remainder.

Syntax:
array gmp_div_qr (resource x, recource y [, int round])

This function combines the action of the two previous functions gmp_div_q () and gmp_div_r (). It returns an array consisting of two elements: with the index [0] - the whole quotient, with the index [1] - the remainder of the division.

```
$ x = gmp_init ("0xf3c3b5");
$ result = gmp_div_qr ($ x, "0xb1");
echo " Integer :" .gmp [strval ($ result [0]);
echo " Remaining :" .gmp [strval ($ result [1]);
```

gmp_mod
Returns the modulus of the remainder of the division.

Syntax:
resource gmp_mod (resource x, recource y)

This function is equivalent to gmp_div_r (), except that it returns an absolute value.

gmp_divexact

Performs **residual** division.

Syntax:
resource gmp_divexact (resource x, recource y)

This function uses the "fine" dividing algorithm. The result is valid only if x is integer divisible by y.

gmp_cmp Compares
two numbers.

Syntax:
int gmp_cmp (resource x, recource y)

The function will return a positive value if x> y; zero if x = y; negative value if x <y.

GMP functions. Maths

gmp_fact
Calculates factorial.

Syntax:
resource gmp_fact (resourse x)

Returns the factorial of the number given in the x parameter.

gmp_sqrt
Calculates the square root.

Syntax:
resource gmp_sqrt (resourse x)

Returns the square root of the number specified in the x parameter.

gmp_sqrtrm
Calculates the square root with remainder.

Syntax:
array gmp_sqrtrm (resourse x)

This function returns an array in which the element with index [0] is the square root of the argument, the element with index [1] is the difference between the argument and element [0] squared.

gmp_perfect_square
Determines if a number is a perfect square.

Syntax:
bool gmp_perfect_square (resourse x)

gmp_perfect_square () will return true if x is the square of an integer. Otherwise it will return false.

gmp_pow
Exponentiation.

Syntax:
resource gmp_pow (resourse x, int y)

This function returns the exponent of x, provided that y is not negative.

```
echo gmp _ pow (2,3); // Prints 8
echo gmp _ pow (0,0); // Output 1
```

gmp_powm

Returns the remainder of a power division.

Syntax:
resource gmp_powm (resourse x, resource y, resource mod)

Returns the remainder of division (x to the power of y) by mod, in case y is positive.

gmp_prob_prime
Check for a "probably" prime number.

Syntax:
int gmp_prob_prime (resourse x [, int reps])

This function will return 0 if x is complex, i.e. which has more than two integer divisors. Will return 1 if x is possibly prime. If it returns 2, then x is probably a prime number.
The reps argument determines the quality of the check. The higher the number, the more accurate the result. It can take values from 5 to 10 (default).

This function uses the Miller-Rabin probabilistic test algorithm.

gmp_gcd
Finds the Greatest Common **Divisor** .

Syntax:
resource gmp_gcd (resourse x, resource y)

Always returns positive.

gmp_gcdext Find the
greatest common factor with factors.

Syntax:
array gmp_gcdext (resourse x, resource y) The gmp_gcdext ( )

function returns an array with the values g, s, t such that x * s + y * t = g = gcd (x, y), where gcd is the greatest total divider.

gmp_invert
Performs modulo inverse.

Syntax:
resource gmp_invert (resourse x, resource y)

The function returns the complement of x to a value divisible by y. If the result cannot be found, it returns false.

gmp_legendre
Returns the Legrange number.

Syntax:
int gmp_legendre (resourse x, resource p)

The function returns the Legrange number. p must be even positive.

gmp_jacobi
Returns the Jacobi number.

Syntax:

int gmp_jacobi (resourse x, resource p)

The function returns the Jacobi number. p must be even positive.

gmp_random
Generates a random number.

Syntax:
resource gmp_random (int limited)

limited sets the length of the generated number. In case the value limited is negative, a negative number is generated.

gmp_popcount
Getting the population.

Syntax:
int gmp_popcount (resource x)

The function returns the numerator of the population.

gmp_hamdist
Calculate distance.

Syntax:
int gmp_hamdist (resource x, resource y)

The function returns the distance between the numbers x and y. The x and y arguments must be non-negative.

GMP functions. Binary operations

gmp_and
Logical AND (AND).

Syntax :
resource gmp_and (resource x, resource y)

gmp_or
Logical OR (OR).

Syntax :
resource gmp_or (resource x, resource y)

gmp_xor
Logical exclusive-OR (XOR).

Syntax :
resource gmp_xor (resource x, resource y)

gmp_setbin
Sets the bit.

Syntax:
resource gmp_setbin (resource & x, int index [, bool set_clear])

Sets the bit at position index in x. The set_clear argument specifies whether to set the bit to 0 or 1 (the default).

gmp_clrbit Clears a

bit.

Syntax:
resource gmp_clrbit (resource & x, int index)

Sets the bit at position index in number x to 0.

gmp_scan0
Searches for bit 0.

Syntax:
0int gmp_scan0 (resource x, int start) The gmp_scan0 ( )

function searches for bits 0 in the number x, starting from the start position, upward. Returns the position of the found bit.

gmp_scan1
Searches for bit 1.

Syntax:
1int gmp_scan1 (resource x, int start) The gmp_scan0 ( )

function searches for bit 1 in the number x, starting from the start position, in the direction of increasing significance of the digits. Returns the position of the found bit.

Working with arrays

Array creation

array
Creates and initializes an array.

Syntax:
array array ([mix ed ...]) The function returns the created array. Indexes and values in an array are separated by the => operator. The index => value pairs are comma separated, they define the index and value. The index can be either numeric or string. In associated arrays, the index always behaves like a string. If the index is not specified, the auto-increment will be substituted (by 1 more), starting from 0. If two elements with the same indices were specified when creating the array, then the last element replaces the first.

```
$ arr = array (// Next, we'll create a two-dimensional array
"fruit" => array ("a" => "orange", "b" => "banan", "c" => "apple"),
// this entry is equivalent to: $ arr ["fruit"] ["a"] = "orange"; etc.
"number" => array (1,2,3,4,5,6),
// this entry is equivalent to the entry: $ arr ["number"] [] = 1; etc.
"hotel" => array ("first", 5 => "second", "third")
);
$ arr = array (1, 1, 1, 1, 2 => 5, 19, 3 => 20);
print_r ($ arr);
// Next, print this array
Array
(
    [0] => 1
    [1] => 1
    [2] => 5
    [3] => 20
    [4] => 19
)
```

```
$ arr = array (1 => "January", "February", "March");
print_r ($ arr);
// printout
Array
(
    [1] => January
    [2] => February
    [3] => March
)
```

range
Populates the list with integers.

Syntax:
list range (int low, int high)

The range ( ) function creates a list filled with integers from *low* to *high*, inclusive. It is convenient to use if we want to quickly generate an array for subsequent traversal through it with a forech loop.

```
$ arr = range (4.9);
// now $ arr = array (4, 5, 6, 7, 8, 9);
```

Sorting arrays

array_reverse Reverse ordering
of array elements.

Syntax:
array array_reverse (array arr);

The **array_reverse ()** function returns an array whose elements are in reverse order relative to the array passed in the parameter. In this case, the connections between keys and values are, of course, not lost. For example, instead of ordering an array in reverse order using **arsort ()** , we can sort it in positive order and then reverse it:

```
$ A = array ("a" => "Zero", "b" => "Weapon", "c" => "Alpha", "d" => "Processor");

asort ($ A);

$ A = array_reverse ($ A);
```

shuffle
Shuffle array elements.

Syntax:
void shuffle (array arr);

The **shuffle ()** function **shuffles the** list passed to it as the first *arr* parameter so that its values are randomly distributed. This changes the array itself and associative arrays are perceived as lists.

```
$ A = array (10,20,30,40,50);
shuffle ($ A);
foreach ($ A as $ v) echo "$ v";
// Prints 10,20,30,40,50 randomly
```

sort
Sorts the array in ascending order.

Syntax:
void sort (array arr [, int sort_flags])

This function is designed to sort lists (lists are arrays whose keys start at 0 and have no gaps) in ascending order.

```
$ A = array ("One", "Two", "Tree", "Four");
sort ($ A);
for ($ i = 0; $ i <count ($ A); $ i ++) echo "$ i: $ A [$ i]";
// prints "0: Four 1: Two 2: Tree 3: One"
```

Any associative array Perceived by this function as a list. That is, after the ordering, the sequence of keys turns into 0,1,2, ..., and the values are redistributed as needed. As you can see, the links between the parameters *key => value are* not saved, moreover, the keys simply disappear, so sorting something other than a list is hardly advisable.

Argument *sort_flags* specifies the following sorting flags:

- SORT_REGULAR - compares items "as is"
- SORT_NUMERIC - compares items as numbers
- SORT_STRING - compares items as strings

rsort
Sort the array in descending order.

Syntax:
void rsort (array arr [, int sort_flags])

Same as **sort ()** , only sorts in descending order.

asort Sorts the
associative array in ascending order.

Syntax:
void asort (array arr [, int sort_flags]);

The **asort ()** function sorts the array specified in its parameter so that its values are in alphabetical (if strings) or ascending (for numbers) order. In this case, the connections between the keys and their corresponding values are preserved, i.e. some *key => value pairs* just "float" up, and some - on the contrary, "down".

```
$ A = array ("a" => "Zero", "b" => "Weapon", "c" => "Alpha", "d" => "Pr ocessor");
asort ($ A);
foreach ($ A as $ k => $ v) echo "$ k => $ v";
// displays "c => Alpha d => Processor b => Weapon a => Zero"
// as you can see, only the order of the key => value pairs has changed
```

By default, the **asort ()** function sorts the array alphabetically. For the values of the *sort_flags*, see the description for the **sort ()** function .

arsort Sorts the
associative array in descending order.

Syntax:
void arsort (array arr [, int sort_flags]);

This function is similar to the **asort ()** function , only it orders the array in descending rather than ascending order.

```
$ arr = array ("d" => "lemon", "a" => "orange", "b" => "banana", "c" => "apple");
arsort ($ arr);
reset ($ arr);
while (list ($ key, $ val) = each ($ arr)) {
  echo "$ key = $ val <BR> l";
}
// will output :
a = orange
d = lemon
b = banana
c = apple
```

ksort
Sort the array in ascending order of keys.

Syntax:
int ksort (array arr [, int sort_flags]);

The function is almost identical to the **asort ()** function , with the difference that sorting is performed not by values, but by keys (in ascending order).

```
$ A = array ("d" => " Zero", "c" => "Weapon", "b" => "Alpha", "a" => "Processor");
ksort ($ A);
for (Reset ($ A); list ($ k, $ v) = each ($ A);) echo "$ k => $ v";
// outputs "a => Processor b => Alpha c => Weapon d => Zero"
```

The *sort_flags* argument *specifies* the sort options.

krsort
Sort the array in descending order of indices.

Syntax:
int krsort (array arr [, int sort_flags]);

This function is similar to the **ksort ()** function , except that it orders the array by keys in reverse order.

natsort
Performs "natural" sorting on an array.

Syntax:
void natsort (array arr);

The **natsort ()** function sorts the array in a human-natural order.

```
$ arr1 = array ("html_12.html", "html_10.html", "html_2.html", "html_1.html");
$ arr2 = $ arr1;
sort ($ arr1);
echo " Standard sort : \ n";
print_r ($ arr1);
natsort ($ arr2 );
echo "Natural sort: \ n"
print _ r ($ arr 2);
```

This example will output the following:

```
Standard sorting:
Array
(
    [0] => html_1.html
    [1] => html_10.html
    [2] => html_12.html
    [3] => html _2. html
)
Natural sorting:
Array
(
    [3] => html_1.html
    [2] => html_2.html
    [1] => html_10.html
    [0] => html_12.html
)
```

uasort
Custom sorting of the associative array.

Syntax:
void uasort (array arr, function cmp_function)

The **uasort ( )** function sorts the *arr* array preserving index associations, using the user-defined function specified by the *cmp_function* argument to compare the element indices .

uksort
Custom sorting of an array by keys.

Syntax:
void uksort (array arr, function cmp_function)

The **uksort ( )** function sorts the *arr* array by index, preserving index associations, using the user-defined function specified in the *cmp_function* argument to compare the element indexes . Two compared indexes of the elements are passed to this function, and it should return a positive or negative number or 0.

Quite often we have to sort something according to a more complex criterion than just alphabetically. For example, suppose *$ Files* contains a list of filenames and subdirectories in the current directory. We may want to print this list not only in lexographic order, but also so that all directories come before the files. In this case, we should use the **uksort ()** function , having previously written a comparison function with two parameters, as required by **uksor t ()** .

```
// This function should compare the values  of $ f1 and $ f2 and return:
// -1 if $ f1 <$ f2,
// 0 if $ f1 == $ f2
// 1 if $ f1> $ f2
// By <and> we mean the following of these names in the output list
function FCmp ($ f 1, $ f 2)
{// Directory always precedes file
  if (is_dir ($ f1) &&! is_dir ($ f2)) return -1;
  // The file always comes after the directory
  if (! is _ dir ($ f 1) && is _ dir ($ f 2)) return 1;
  // Otherwise compare lexicographically
  if ($ f1 <$ f2) return -1; elseif ($ f1> $ f2) return 1; else return 0;
}
// Let $ Files contain an array with keys - file names
// in the current directory. Let's sort it out.
uksort ($ Files, "FCmp"); // pass the sort function "by reference"
```

usort Uses an
array sort.

Syntax:
void usort (array arr, function cmp_function)

The **uso rt ( )** function sorts the *arr* array preserving index associations, using the user-defined function specified in the *cmp_function* argument to compare element indices . Two compared indexes of the elements are passed to this function, and it must return a positive or negative number or 0.

This function is, as it were, a "hybrid" of the **uasort ()** and **sort ()** functions . It differs from **sort ()** in that the comparison criterion is provided by a user-defined function. And from **uasort ()** - in that it does not preserve relationships between keys and values, and therefore is only suitable for sorting lists.

```
function FCmp ($ a, $ b) {return strcmp ($ a, $ b); }
$ A = array ("One", "Two", "Three", "Four");
usort ($ A);
for ($ i = 0; $ i <count ($ A); $ i ++) echo "$ i: $ A [$ i]";
// prints "0: Four 1: One 2: Three 3: Two"
```

An example of a one-dimensional array:

```
function cmp ($ a , $ b ) {
  if ($ a == $ b) return 0;
  return ($ a> $ b)? -eleven;
}
$ a = array (3,2,5,6,1);
```

```
usort ($ a, "cmp");
while (list ($ key, $ val) = each ($ a)) {
  echo "$ key : $ val \ n ";
}
```
When executed, it will print:
```
0: 6
fifteen
2: 3
3: 2
4: 1
```
An example of a multidimensional array:
```
function cmp ($ a , $ b ) {
  return strcmp ($ a ["fruit"], $ b ["fruit"]);
};
$ fruit [0] ["fruit"] = "lemons";
$ fruit [1] ["fruit"] = "apples";
$ fruit [2] ["fruit"] = "grapes";

usort ($ fruit, "cmp");

while (list ($ key, $ val) = each ($ fruit)) {
  echo "\ $ fruit [$ key]:". $ val ["fruit"]. "\ n";
}
```
When sorting multidimensional arrays, $ a and $ b contains references to the first index of the array.
It will print:
```
$ fruit [0]: apples
$ fruit [1]: grapes
$ fruit [2]: lemons
```

array_multisort
Sorting relational arrays.

Syntax :
bool array_multisort (array ar1, [, mixed o1 [, mixed t1 ... [, array ...]]])

The **array_multisort ()** function sorts multidimensional arrays preserving the index association, returning true on no error.

The original arrays are treated as columns of a table sorted row by row. Therefore, the arrays must have the same number of elements, and the relationship between them, as in the rows of a table, is preserved. The first arrays take priority of sorting. Sort flags can be specified for each array, and their effect only applies to the array after which they are specified.

Sort order flags (*ox* arguments):
- SORT_ASC - sort in ascending order (default)
- SORT_DESC - sort in descending order. Sort type flags (*tx* arguments):
- SORT_REGULAR - compare items as they are (default)
- SORT_NUMERIC - compare items as numbers
- SORT_STRING - compare items as strings

```
ar1 = array ("10", 100, 100, "a");
ar2 = array (1, 3, "2", 1);
array_multisort ($ ar1, $ ar2);
  // $ ar1 = array ("10", "a", 100, 100);
  // $ ar2 = array (1, 1, "2", 4);
```
The elements of the second array corresponding to the same elements (100 and 100) of the first array are also sorted.
```
$ ar = array (array ("10", 100, 100, " a"), array (1, 3, "2", 1));
array_multisort ($ ar [0], SORT_ASC, SORT_STRING,
                $ ar [1], $ SORT_NUMERIC, SORT_DESC);
```
$ ar [0] = ("10", 100, 100, "a") - sorted as strings in ascending order $ ar [1] = (1, 3, "2", 1) - sorted as

numbers in descending order .

Array cursor

reset
Resets the array cursor.

Syntax:
mixed reset (array arr);

The **reset ()** function sets the internal cursor of *arr* to the beginning and returns the value of the starting element.

end
Moves the cursor to the end of the array.

Syntax:
mixed end (array arr);

The **end ()** function sets the inner cursor of *arr* to the last element and returns the value of the starting element.

next
Moves the cursor forward.

Syntax:
mixed next (array arr);

The **next ()** function returns the value of the element the cursor is currently on and advances the array cursor to the next element. Returns false if there are no more items left.

Also, false is returned if an element with an empty value is encountered, therefore, to work correctly with an array containing empty elements, it is better to use the **each ()** function .

prev
Moves the cursor back.

Syntax:
mixed prev (array arr);

The **prev ()** function returns the value of the element the cursor is currently on and moves the array cursor to the previous element. Returns false if there are no more items left.

Also, false is returned if an element with an empty value is encountered, therefore, to work correctly with an array containing empty elements, it is better to use the **each ()** function .

current
The definition of the current element of the array.

Syntax:
mixed current (array arr);

The **current ()** function returns the value of the element on which the array cursor is currently located, without moving the cursor.

The function will return false if the cursor is outside the array or the array has no elements.

pos
Determination of the current array element.

Syntax:
mixed pos (array arr);

This function is synonymous with the **current ()** function .

key
The function returns the index of the current array element.

Syntax :
mixed key (array arr);

each
Get the current element of the array.

Syntax:
array each (array arr);

The **each ()** function returns the "index and value" pair of the current array element pointed to by the inner cursor in an array and advances the array cursor to the next element. The returned array has four elements:

```
[0] => index
[1] => "value"
[key] => index
[value] => "value"
```

The function returns false if the cursor has reached the end of the array.

```
$ foo = array ("bob", "fred", "jussi", "joini", "egon", "marliese");
$ bar = each ($ foo);
// now $ bar = (0 => 0, 1 => "bob", key => 0, value => "bob")
```

Typically the **each ()** function is used in conjunction with **list ()** to iterate over the elements of an array:

```
reset ($ HTTP_POST_VARS);
while (list ($ ke y, $ val) = each ($ HTTP_POST_VARS)) {
  echo "$ key =% val <BR>";
}
```

array_walk
Applying a function to array elements.

Syntax:
int array_walk (array arr string func, mixed userdata);

The **array_walk ()** function applies the custom *func* function to each element in the *arr* array . Three or two (if no *userdata* argument is specified) arguments are passed to a custom function : the value of the current element, its index, and the *userdata* argument .

If *func* requires more than three arguments, a warning will be issued each time it is called. To block these warnings from issuing, put an @ sign in front of **array_walk ()** or use **error_reporting ()** .

The *func* function will get the values and indices of the *arr* array by value, i.e. will not be able to make changes to it. If necessary, pass the *arr* argument by reference, prefixing its name with "&", and then all changes will be reflected in the array.

In PHP 4, you must explicitly call the **reset ()** function to set the inner cursor to the first element.

```
$ v = array ("d" => "A1", "a" => "B2", "b" => "C3", "c" => "D4");
function test_alter (& $ item1, $ key, $ prefix) {// by reference
  $ item1 = "$ prefix $ item1";
};

function test_print ($ item2, $ key) {
  echo "$ key. $ item2 <BR>";
};
```

```
array_walk ($ v, "test_print") ;
reset ($ v);
array_walk ($ v, "test_alter");
reset ($ v);
array_walk ($ v, "test_print");
```

Keys and Values

array_flip Swaps the
indices and values of an array.

Syntax:
array array_flip (array arr)

This function "iterates" through an array and swaps its keys and values. The original arr array is unchanged, and the resulting array is simply returned. Of course, if there were several elements with the same values in the array, only the last one will be taken into account:

```
$ A = array ("a" => "aaa", "b" => "a aa", "c" => "ccc");

$ A = array_flip ($ A);

// Now $ A === array ("aaa" => "b", "ccc" => "c");
```

array_keys
Returns a list of array keys.

Syntax:
list array_keys (array arr [, mixed search_value])

The function returns a list containing all the keys of the array *a rr* . If the optional parameter *search_value is given* , then it will return only those keys that match the *search_value* .

```
$ arr = array (0 => 100, "color" => "red", 15);
array_keys ($ arr); // will return array (0, "color", 1)

$ arr = array ("blue", "red", "green", "blue", "blue");
array_keys ($ arr, "blue"); // will return array (0, 3, 4)
```

array_values
Remove associative array indices.

Syntax:
list array_values (array arr)

The **array_values (**) function returns a list of all values in the associative array *arr* , i.e. turns an associative array into a simple (scalar) one.

```
$ arr = array ("size" => "XL", "color" => "gold");
array_values  ($ arr);
// returns array ("XL", "gold")
```

Obviously, this action is useless for lists, but sometimes it is justified for hashes.

in_ array Checks the array
for a value.

Syntax:
bool in_array (mixed val, array arr)

The **in_array (**) function **will** return true if the *arr* array contains an element with the value *var* .

```
$ arr = array ("1", "2", "tree");
if (in_array ["2", $ arr)) e cho "2 is ";
```

array_count_values
Returns the number of identical values in an array.

Syntax:
array array_count_values (array arr)

This function counts how many times each value occurs in the *arr* array , and returns an associative array with keys - the elements of the array and values - the number of times those elements are repeated . In other words, the **array_count_values ()** function counts the frequency of occurrence of values in the *arr* array .

```
$ LIst = array (1, "hello", 1, "world", "hello");
array_count_values  ($ array);
// returns array (1 => 2, "hello" => 2, "world" => 1)
```

sizeof
Returns the number of elements in the array.

Syntax:
int sizeof (array arr)

The **sizeof (**) function returns the number of elements in the *arr* array, similar to **count ()** .

count
Returns the number of elements in an array or object.

Syntax:
int count (mixed var)

The **count (**) function returns the number of elements in an array or *var* object . If *var* is a scalar variable, then the function returns 1 if such a variable exists, or 0 if there is no such variable.
It should be noted that 0 is returned even when an array is specified that does not contain elements.
It is better to use the **isset ()** function to check if a variable exists .

```
$ a [0] = 1;
$ a [1] = 3;
$ a [2] = 5;
$ result = count ($ a) // will return 3
```

array _sum
Returns the sum of all the elements in an array.

Syntax:
mixed array_sum (array arr [, int num_req])

This function will return the sum of all numeric elements in the array. The type of the values in the array determines the type of the returned number (integer or float).

```
$ arr = array (2,4,6,7);
echo " Amount :" .array_sum ($ arr);
// will display Amount: 19
```

array_rand
Produces a random selection of array indices.

Syntax:
mixed array_rand (array arr [, int num_req])

The **array_rand ()** function returns randomly selected indices of the elements of the array *arr in an* array . The *num_req* argument specifies the number of indexes to return. If one element is selected, then not an array is returned, but a value.

```
srand (( double ) microtime () * 1000000);
```

```
// here we have initialized the random number generator
$ arr = array ("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$ rand_keys = array_rand ($ arr, 2);
echo $ arr [$ rand_key [0]]. "<BR>";
echo $ arr [$ rand_key [1]]. "<BR>";
```

Complex replacement in a string

strtr
Complex replacement in a string.

Syntax:
string strtr (string str, string from, string to)
string strtr (string str, array from)

In the first case, the **strtr ()** function returns a string *str* , in which each character present in the string *from* is replaced with the corresponding string *to* ... If the strings *from* and *to are of* different lengths, then the extra trailing characters of the long string are ignored.

In the second case, the **strtr ()** function returns a string in which fragments of the *str* string are replaced with the values of the *from* array elements corresponding to the indices . In this case, the function tries to replace the largest fragments of the original string first and does not perform replacement in the already modified parts of the string. Thus, now we can perform several replacements at once:

```
$ Subs = array (
  "<name>" => "Larry",
  "<time>" => date ("dmY")
);
$ st = "Hello < name >! It's < time ";
echo strtr ($ st , $ Subs );
```

And here's how you can "undo" the action of the **HtmlSpecialChars ()** function :

```
$ Trans = array_flip (get_html_translation_table ());
$ st = strtr ($ st , $ Trans );
```

As a result, we will get the original string from a string in which all special characters are replaced by their HTML equivalents.

Working with multiple arrays

array_diff
Define an exceptional intersection of arrays.

Syntax :
array array_diff (array arr1, array arr2 [, array ...])

This function returns an array that contains values that are only in the array *arr1* (and not any others). However, the indexes are not saved.

```
$ arr 1 = array (" a " => " green ", " red ", " blue ");
$ arr2 = array ("b" => "green", "yellow", "red");
$ result = array_diff ($ arr1, $ arr2);
// $ result = array ("blue")
```

array_intersect
Define the inclusive intersection of arrays.

Syntax:
array array _ intersect (array arr 1, array arr 2 [, array ...]) The **array_intersect ()** function returns an array that contains the values of the array *arr1* found in all other arrays. However, the indexes are not saved.

```
$ arr 1 = array (" a " => " green ", " red ", " blue ");
$ arr2 = array ("b" => "green", "yellow", "red");
$ result = array_intersect ($ arr1, $ arr2);
```

```
// $ result = array ("a" => "green", "red")
```

array_merge
Merge arrays . **Syntax :** array array_merge (array arr1, array arr2 [, array ...]) The **array_merge ()** function is designed to eliminate all the drawbacks inherent in the + operator for array merging. Namely, it merges the arrays listed in its arguments into one large array and returns the result. If the arrays contain the same keys, the result is filled with the *key => value* pair from the array located to the right in the list of arguments. However, this does not affect numeric keys: elements with such keys are placed at the end of the resulting array anyway.

```
$ L1 = array (10,20,30);
$ L2 = array (100,200,300);
$ L = array_merge ($ L1, $ L2);
// now $ L === array (10,20,30,100,200,300);
```

array_merge_recursive
Combining complex arrays.

Syntax :
array array_merge_recu rsive (array arr1, array arr2 [, array ...])

The **array_merge_recursive ()** function strongly resembles the **array_merge ()** function with the addition that it can work with multidimensional and tree-like arrays, and elements with the same string indices are converted into subarrays. For numeric indices, the function behaves like **array_merge ()** .

```
$ arr 1 = array (" color " => array (" favorite " => " red "), 5);
$ arr2 = array (10, "color" => array ("favorite" => "green"), "blue");
$ result = array_merge_recursive ($ arr1, $ arr2);
// $ result = array ("color" => array (
// "favorite" => array ("red", "green"),
// "blue"), 5, 10)
```

Getting and removing part of an array

array_slice
Get part of an array.

Syntax:
array array_slice (array arr, int offset [, int len])

This function returns the part of the associative array *arr* , starting at the element with offset (number) *offset* from the beginning and length *len* (if the last parameter is not specified, to the end of the array).

The *offset* and *len* parameters follow exactly the same rules as the analogous parameters in the **substr ()** function . Namely, if *offset> 0* , then the sequence will start from the element at the *offset* position from the beginning of the array, and if <0, then the sequence will start from the end of the array. It should be noted that the first element has a zero position, and the last one (-1).
If you specify *length* > 0, then this is the number of elements returned in the array, and if *length* <0, then this is the position of the last returned element in the *arr* array from its end.

```
$ input = array ("a", "b", "c", "e");
$ output = arr ay_slice ($ input, 2); // "c", "d", "e"
$ output = array_slice ($ input, 2, -1); // "c", "d"
$ output = array_slice ($ input, -2, 1); // "d"
$ output = array_slice ($ input, 0, 3); // "a", "b", "c"
```

array_splice
Removes part of an array or replaces it with part of another array.

Syntax:
array array_splice (array arr, int offset [, int len] [, int repl])

This function, like **array_slice ()** , returns a subarray *arr* starting at the *offset* index of the maximum length *len* , but at the same time it does and other useful action. Namely, it replaces the elements just specified with what is in the *repl* array (or simply removes if *repl is* not specified). If *offset> 0* , then the sequence will start from the element at *offset* from the beginning of the array, and if <0, then the sequence will start from the end of the array. It should be noted that the first element has a zero position, and the last one (-1).
If you specify *length* > 0, then this is the number of elements returned in the array, and if *length* <0, then this is the position of the last returned element in the *arr* array from its end.

```
$ input = array ("red", "green", "blue", "yellow");
array_splice ($ input, 2);
// Now $ input === array ("red", "green")
array_splice ($ input, 1, -1);
// Now $ input === array ("red", "yellow")
array_splice ($ input, -1, 1 , array ("black", "maroon"));
// Now $ input === array ("red", "green", "blue", "black",
        "maroon")
array_splice ($ input, 1, count ($ input), "orange");
// Now $ input === array ("red", "orange")
```

The last example shows that as the *rep l* parameter, we can also specify an ordinary, string value, and not an array of one element.

Inserting / removing elements

array_pad
Adds multiple elements to an array.

Syntax:
array array_pad (array input, int pad_size, mixed pad_value)

The **array_pad (**) function returns a copy of the *input* array to which elements with *pad_values* have been added , so that the number of elements in the resulting array is equal to *pad_size* .
If *pad_size* > 0, then elements will be added to the right, and if <0, then to the left.
If the *pad_size* value *is* less than the elements in the original *input* array , then no addition will occur, and the function will return the original *input* array .

```
$ arr = array (12, 10, 4);
$ result = array_pad ($ arr, 5, 0);
// $ result = array (12, 10, 4, 0, 0);
$ result = array_pad ($ arr, -7 , -1);
// $ result = array (-1, -1, -1, -1, 12, 10, 4)
$ result = array_pad ($ arr, 2, "noop");
// won't add
```

array_pop
Retrieves and removes the last elements of an array.

Syntax:
mixed array_pop (array arr);

The **array_pop ()** function **pops** an item from the "top" of the stack (that is, it takes the last item in the list) and returns it, then removing it from *arr* . With this function, we can build stack-like structures. If the *arr* array was empty, the function returns an empty string.

```
$ stack = array (" orange ", " app le ", " raspberry ");
$ fruits = array_pop ($ stack);
// $ fruit = "raspberry"
// $ stack = array ("orange", "apple")
```

array_push
Adds elements to the end of an array.

Syntax :
int array_push (of array arr, mixed var1 [, var2 mixed, ..])

This function adds to the array *arr* elements *var1, var2* , etc. It assigns them numerical indices - just like it does for the standard []. If you only need to add one element, it is probably easier to use this operator:

```
array_push ($ Arr, 1000); // call the function
$ Arr [] = 100; // the same, but shorter
```

Note that **array_push ()** takes an array like a stack and always adds elements to its end.

array_shift Extracts
and removes the first element of an array.

Syntax:
mixed array_shift (array arr)

This function retrieves the first element of the *arr* array and returns it. It strongly resembles **array_pop ()** , but it only receives the initial, not the final element, and also produces a rather strong "shake-up" of the entire array: after all, when extracting the first element, you have to adjust all the numeric indices of all remaining elements, since all subsequent elements of the array are shifted one position forward.

```
$ ar = array ("- v", "-f");
$ opt = array (_shift ($ arr);
// now $ arr = array ("- f") and $ opt = "-v"
```

array_unshift
Adds elements to the beginning of an array.

Syntax :
int array_unshift (list arr, mixed var1 [, mixed var2, ...])

The function is very similar to **array_push** , but adds the listed elements not to the end, but to the beginning of the array. In this case, the order of *var1, var2* , etc. remains the same, i.e. the elements seem to "slide" into the list on the left. New list items are assigned numeric indices, as usual, starting at 0; in this case, all keys of the old array elements, which were also numeric, change (most often they increase by the number of inserted values). The function returns the new size of the array.

```
$ A = array (10, " a " => 20.30);
array _ unshift ($ A , "!", "?");
// now $ A === array (0 => "!", 1 => "?", 2 => 10, a => 20, 3 => 30)
```

array_unique
Creates an array of unique values only.

Syntax:
array array_unique (array arr)

The **array_unique (**) function returns an array composed of all the unique values in *arr* along with their keys, by removing all duplicate values. The first encountered *key => value* pairs are placed in the resulting array . Indexes are saved.

```
$ input = array ("a" => "green", "red", "b" =>
        "green", "blue", "red");
$ result = array_unique ($ input);
// now $ result === ("a" => "green", "red", "blue");
```

Variables and Arrays

list
Puts array elements into variables.

Syntax:
list () is a language construct (like **array ()**). It assigns to the listed variables the values of the array elements, the first variable being assigned the first array element, the second variable the second element, and so on.

compact
Packs variables from the current context into an array.

Syntax :
array compact (mixed varname1 [, mixed $ varname2, ...])

The **compact (**) function packs variables from the current context (global or function context) into an array, specified by their names in *varname1*, *$ varname2* , etc. In this case, pairs are formed in the array with keys equal to the contents of *varnameN* and the values of the corresponding variables.
The number of arguments can be undefined.
If the argument contains the name of a variable that does not exist, it is skipped. This function is the opposite of **extract ()** .

```
$ a = " Test string ";
$ b = " Some text ";
$ A = compact ("a", "b");
// now $ A === array ("a" => "Test string", "b" => "Some text")
```

Why then are the function parameters labeled mixed? The point is that they can be not only strings, but also lists of strings. In this case, the function of the latter iterates over all the elements of this list, and packs those variables from the current context whose names it has encountered. Moreover - these lists can, in turn, also contain lists of strings, etc. True, the latter is rarely used.

```
$ a = "Test";
$ b = "Text";
$ c = "CCC";
$ d = "DDD";
$ List = array ("b", array ("c", "d"));
$ A = compact ("a", $ List);
// now $ A === array ("a" => "Test", "b" => "Text",
        "c" => "CCC", "d" => "DDD")
```

extract -
exports array elements into variables.

Syntax:
void extract (array arr [, int extract_type] [, string prefix])

This function performs the exact opposite of **compact ()** . Namely, it receives the *arr* array in the parameters and turns each of its *key => value pairs* into a variable of the current context.

The *extract_type* parameter *specifies* what to do if a variable already exists in the current context with the same name as the next key in *arr* . It can be one of the constants listed in the following table:

Behavior functions **extract** in the case of coincidence of variables	
EXTR_OVERWRITE	Overwrite existing variable.
EXTR_SKIP	Do not overwrite a variable if it already exists.
EXTR_PR EFIX_SAME	If names match, create a variable with a name preceded by the prefix from *$ prefix* .
EXTR_PREFIX_ALL	Always prefix generated variable names with *$ prefix* .

The default is *EXTR_OVERWRITE* , i.e. variables are overwritten.

```
// Make all environment variables global
extract ($ HTTP _ ENV _ VARS );
// The same, but with the E_ prefix
extract ($ HTTP_ENV_VARS, EXTR_PREFIX_ALL, "E_");
echo $ E _ COMSPEC ;
```

```
// Outputs the COMSPEC environment variable
```

The *prefix* parameter only makes sense to specify when you are using *EXTR_PREFIX_SAME* or *EXTR_PREFIX_ALL modes* .

String functions

Functions for working with single characters

chr
Returns one character with a specific code.

Syntax :
string chr (int ascii)

Returns a one-character string with *$ code* . This function is useful for inserting any non-printable characters into a string, such as a zero code or a page feed, or when working with binary files.

```
<?
// First, create an array of what we are going to output,
// not caring about the formatting (design) of the information
for ($ i = 0, $ x = 0; $ x <16; $ x ++) {
  for ($ y = 0; $ y <16; $ y ++) {
    $ Chars [$ x] [$ y] = array ($ i, chr ($ i));
    $ i ++;
  }
}
// Now we display the accumulated information using the ideology
// insert sections of code into an HTML document
?>

<table border = 1 cellpadding = 1 cellspacing = 0>
<? for ($ y = 0; $ y <16; $ y ++) {?>
  <tr>
  <? for ($ x = 0; $ x <16; $ x ++) {?>
    <td>
      <? = $ Chars [$ x] [$ y] [0]?>:
      <b><tt><?=$Chars [$x] [[]] [[]] [[]] </b>
    </td>
  <?}?>
  </tr>
<?}?>
</table>
```

ord
Returns the ascii character code.

Syntax:
int ord (string str)

This function returns the ASCII code of the first character of string *str* .
For example, ord (chr ($ n)) is always $ n - of course, if $ n is between zero and 255.

Cut off whitespace functions

trim
Removes leading and trailing whitespace from the specified string.

Syntax:
string trim (string str)

Returns a copy of *str* , with only leading and trailing whitespace removed. Whitespace means "\ n", "\ r", "\ t", "\ v", "\ 0" and a space.
For example, calling trim ("test \ n") will return the string "test".

ltrim
Removes leading whitespace from the specified string.

Syntax:
string ltrim (string str)

Same as **trim ()** , but only removes leading whitespace characters ("\ n", "\ r", "\ t", "\ v", "\ 0" and space), but does not touch the terminal.

rtrim
Removes trailing whitespace from the specified string.

Syntax:
string rtrim (string str)

Same as **trim ()** , except that it only removes trailing whitespace characters ("\ n", "\ r", "\ t", "\ v", "\ 0" and space), but does not touch the initial ones.
This function is synonymous with **chop ()** .

chop
Removes trailing whitespace from the specified string.

Syntax:
string chop (string str)

Removes only trailing spaces, does not touch leading ones.

Search in text

strchr Finds the
first occurrence of a character in a string.

Syntax:
string strchr (string haystack, string needle)

This function works identically to the **strstr ()** function .

strstr Finds the
first occurrence of a substring in a string.

Syntax:
string strstr (string haystack, string needle)

The **strstr (**) function returns the portion of the string specified by the *haystack* parameter , from the first portion specified in the *needle* parameter to the end.
Returns false on failure.
This function is case sensitive.
If *needle is* not a string, then the value is converted to an integer and used as the code of the desired character.

```
$ email = "mailname@mail.ru";
$ domain = strstr ($ email, "@");
// or
$ domain = strstr ($ email, ord ("@"))
echo $ domain;
// displays @ mail.ru
```

stristr Finds the
first occurrence of a substring, case **insensitive** .

Syntax:
string stristr (string haystack, string needle)

The **stristr (**) function returns the portion of the string specified by the *haystack* parameter , from the first portion specified in the *needle* parameter to the end.
Returns false on failure.
This function is case insensitive.
If *needle is* not a string, then the value is converted to an integer and used as the code of the desired character.

strrchr
Find the last occurrence of a substring.

Syntax:
string strrchr (string haystack, string needle)

The **strrchr (**) function returns the portion of the string specified by the *haystack* parameter , from the last portion specified in the *needle* parameter to the end.
Returns false on failure.
This function is case sensitive.
If *needle is* not a string, then the value is converted to an integer and used as the code of the desired character.

```
// get the last directory in $ PATH
$ dir = substr ( strrchr ($ PATH , ":"), 1);
// and here we get everything after the last line feed
$ text = "text 1 \ nText2 \ nText3";
echo substr (strrchr ($ text, 10), 1);
```

strpos
Finds the position of the first occurrence of a substring in the given string.

Syntax:
int strpos (string where, string what [, int fromwhere])

The **strpos (**) function tries to find the substring *what* in the string *were* and, if successful, returns the position (index) of this substring in the string. The first character of the string has index 0. The optional parameter *fromwhere* can be specified if you want to search not from the beginning of the string, but from some other position. In this case, this position should be passed to *fromwhere* . If the substring could not be found, the function returns false. If the *what* parameter is not a string, in this case its value is converted to an integer and used as the code of the desired character.

```
if (strpos ($ text, " a") === false) echo " Not found !";
// Check: three equal signs
```

strrpos
Finds the last position in the given string at which the given chunk is.

Syntax:
int strrpos (string where, string what)

This function searches in the *where* string for the last position in which the character *what* was encountered (if *what* is a string of several characters, then only the first one is detected, the rest play no role).
If the required character is the first in the string or it is not at all, the function will return 0.
If the required character is not found, it returns false.

substr_count
Finds the number of occurrences of a fragment in a string.

Syntax:
int substr_count (string where, string what)

The **substr_count ()** function returns the number of *what* fragments present in the *where* string .

```
e cho substr_count ("www.spravkaweb.ru", ".");
// Prints 3
```

strspn
Determines the presence of leading characters in the string.

Syntax:
int strspn (string str1, string str2)

The **strspn (**) function returns the length of the initial portion of *str1* , which consists entirely of the characters in *str2* .

```
echo strspn ("www.spravkaweb.ru", "abc");
// Prints 3
```

strcspn
Determines the absence of leading characters in the string.

Syntax:
int strcspn (string str1, string str2)

The **strspn (**) function returns the length of the initial piece of string *str1* , which is completely different from the characters in string *str2* .

Comparison functions

strcmp
Compares strings.

Syntax:
int strcmp (string str1, string str2)

This function compares two strings character-by-character (more precisely, byte-wise) and returns:
0 - if the strings are exactly the same;
-1 - if string *str1 is* lexicographically less than *str2* ;
1 - if, on the contrary, *str1 is* "greater than" *str2* .
Since the comparison is byte-wise, the case of characters affects the results of comparisons.

strncmp
Compares the beginning of lines.

Syntax:
int strncmp (string str1, string str2, int len)

This function differs from **strcmp ()** in that it does not compare the entire word, but the first *len* bytes. If *len is* less than the length of the smallest of the strings, then the entire strings are compared.
This function compares two strings character by character (more precisely, byte bytes) and returns:
0 - if the strings are exactly the same;

-1 - if string *str1 is* lexicographically less than *str2* ;
1 - if, on the contrary, *str1 is* "greater than" *str2* .
Since the comparison is byte-wise, the case of characters affects the results of comparisons.

strcasecmp
Compares strings case insensitive.

Syntax:
int strcasecmp (string str1, string str2)

Same as **strcmp ()** , except that it is not case sensitive.

```
$ str1 = " Hello !";
$ str2 = " hello !";
if (! strcesecmp ($ str1, $ str2))
  echo "$ str1 == $ str2 for case insensitive string comparison";
```

strncasecmp
Compares the beginning of strings, case insensitive.

Syntax:
int strncasecmp (string str1, string str2, int len)

The **strncasecmp ()** function is a combination of the **strcasecmp ()** and **strncmp ()** functions .

strnatcmp
Performs "natural" string comparisons.

Syntax:
int strnatcmp (string str1, string str2)

This function simulates a string comparison that a human would use.

```
$ arr1 = $ arr2 = array ("img12.png", "im g10.png", "img2.png", "img1.png");
echo " Normal sort \ n";
usort ($ arr1, "strcmp");
print _ r ($ arr 1);
echo "\ nNatural sorting \ n";
usort ($ arr2, "strnatcmp");
print_r ($ arr2);
```

This script will display the following:

```
Normal sort
Array
(
   [0] => img1.png
   [1] => img10.png
   [2] => img12.png
   [3] => img 2. png
)

Natural sorting
Array
(
   [0] => img 1. png
   [1] => img 2. png
   [2] => img 10. png
   [3] => img12.png
)
```

strnatcasecmp
Performs natural case-insensitive string comparisons.

Syntax:
int strnatcasecmp (string str1, string str2)

Same as **strnatcmp ()** , but ignores case.

similar_text
Determines the similarity of two strings.

Syntax:
int similar_text (string firsrt, string second [, double percent])

The **similar_text ()** function calculates the similarity of two strings using the algorithm described by Oliver (Oliver [1993]). But instead of the stack (as in Oliver's pseudocode), it uses recursive calls.
The complexity of the algorithm makes the function slow and its speed is proportional to (N ^ 3), where N is the length of the largest string.
The function returns the number of characters matched in both strings. When passing a third optional parameter by reference, it stores the percentage of lines that match.

levenshtein
Determination of Levenshtein difference between two strings.

Syntax:
int levenshtein (string str1, string str2)
int levenshtein (string str1, string str2, int cost_ins, int cost_rep, int cost_del)
int levenshtein (string str1, string str2, function cost)

"Levenshtein difference" is the minimum number of characters which would have to be replaced, inserted, or deleted in order to turn *str1* into *str2* . The complexity of the algorithm is proportional to the product of the lengths of *str1* and *str2* , which makes the function faster than *similar_text ()* .

The first form of the function returns the number of operations required on string characters to transform *str1* to *str2* .

The second frame has three additional parameters: the cost of an insert, replace, and delete operation, which makes it more computationally adaptable, but slower. The integral indicator of transformation complexity is returned.

The third option allows you to specify the function used to calculate the complexity of the transformation. The *cost* function is called with the following arguments:
- applied operation (insert, change, delete): "I * quot ;," R "," D ";
- the actual character of the first line
- the actual character of the second line
- line position 1
- line position 2
- remaining length of line 1
- remaining length of string 2

The called function will have to return the cost of this operation.
If one of the strings is more than 255 characters **long, levenshtein ()** returns -1, but that length is more than sufficient.

Formatting and Outputting Strings

print
Prints a string, variable value, or expression.

Syntax:
print (string arg)

The **print (**) function prints the argument *arg* , which can be a variable or an expression.

echo
Prints one or more values.

Syntax :
echo (string arg1, string [argn] ...)

The **echo ()** function outputs the values of the listed parameters.
echo () is actually a language construct, so no parentheses are required for it, even if multiple arguments are used.

```
echo "Line break,
existing in the code is saved
and is used when displaying ".
"to avoid this use".
"concatenation operator";
```

printf
Output a formatted string.

Syntax:
int printf (string format [, mixed args, ...]);

Does the same thing as **sprintf ()** , only the resulting string is not returned, but sent to the user's browser.

sprintf
Performs variable substitution string formatting.

Syntax: sprintf ($ format [, args , ...])
This function returns a string based on a format string containing some special characters that will be subsequently replaced with the values of the corresponding variables from the argument list.

The *$ format format* string can include formatting commands preceded by a% character. All other characters are copied to the output string as is. Each format specifier (that is, the% character followed by the commands) matches one and only one parameter, specified after the *$ format* parameter . If you need to put% in the text as a regular character, you need to double it:

echo spr intf ("The percentage was% d %%", $ percentage);

Each format specifier has a maximum of five elements (in the order they appear after the% character):
- An optional field size specifier that specifies how many characters will be allocated for the displayed value. Placeholder characters (if the value is smaller than the size of the field for its output) can be a space or 0, the default is a space. You can specify any other filler character if you specify it in the format string, preceded by a phpostrophe.
- An optional alignment specifier that determines whether the result will be right-aligned or left-aligned. The default alignment is right, but you can also specify left alignment by specifying the - (minus) symbol.
- An optional number specifying the size of the field for outputting the value. If the result does not fit into the field, then it will "go out" beyond the edges of this field, but will not be truncated.
- An optional number, preceded by a dot ".", Specifying how many decimal places will be in the resulting string. This specifier is taken into account only if a floating point number is being output, otherwise it is ignored.
- Finally, the required (note - the only required!) Value type specifier, which will be placed in the output line:
- b - the next argument from the list is displayed as a binary integer
- c - displays a symbol with the code specified in the argument
- d - integer
- f - floating point number
- o - octal integer
- s - character string

- x - hexadecimal integer with small letters az
- X is a hexadecimal integer with capital letters AZ

Here's how you can specify the precision of floating point numbers:

```
$ money1 = 68.75;
$ money2 = 54.35;
$ money = $ money1 + $ money2;
// echo $ money will print "123.1" ...
$ formatted = sprintf ("% 01.2f", $ money);
// echo $ formatted will print "123.10"!
```

Here is an example of outputting an integer, preceded by the required number of zeros:

```
$ i sodate = sprintf ("% 04d-% 02d-% 02d", $ year, $ month, $ day);
```

sscanf
Interprets the string according to the format and sets the values to variables.

Syntax:
mixed sscanf (string str , string format [, string var 1 ...]) The **sscanf ()** function is the opposite of the **printf ()** function . It interprets the string *str* according to *format* , similar to the **printf ()** specification . If only two arguments are specified, the resulting values are returned in an array.

```
// get serial number
$ serial = sscanf ("SN / 235-0001", "SN /% 3d-% 4d");
echo $ serial [0] * 10000 + $ serial [1]; // outputs: 2350001
// and production date
$ date = " January 01 2000 ";
list ($ month, $ day, $ year) = sscanf ($ date, "% s% d% d");
echo " Date : $ year -". substr ($ month, 0,3). "- $ day \ n";
// outputs: 2000-Jan-01
```

If additional optional parameters are specified (they must be passed by reference), the function returns their number. Variables that do not receive values are not included in the return value.

```
// generate XML write from string
$ auth = "765 \ tLewis Carr oll ";
$ n = sscan ($ auth, "% d \ t% s% s", & $ id, & $ first, & $ last);
echo "<author id =" $ id ">
  <firstname> $ first </firrstname>
  <surname> $ last </surname>
</author> \ n ";
```

Composing / splitting lines

substr
Returns a portion of a string with a specific length.

Syntax:
string substr (string str, int start [, int length])

Returns the portion of string *str* starting at position *start* and length *length* . If *length is* not specified, then the substring from *start* to the end of *str is* assumed . If *start is* greater than the length of the string, or if *length* is zero, then an empty substring is returned.

However, this function can also do some pretty useful things. For example, if we pass a negative number to *start* , then it will be considered that this number is the substring index, but only counted from the end of *str* (for example, -1 means "starts from the last character of the string").
Parameter *the length* , if present, can also be negative. In this case, the last character of the returned substring will be the character from *str* with the index *length* , determined from the end of the string.

```
$ str = substr ("abcdef", 1); // will return "bcdef"
$ str = substr ("abcdef", 1, 3); // will return "bcd"
$ str = substr ("abcdef", -1); // will return "f"
$ str = substr ("abcdef", -2); // will return "ef"
$ str = substr ("abcdef", -3, 1); // will return "d"
$ str = substr ("abcdef", 1, -1); // will return "bcde"
```

str_repeat
Repeats a string a specified number of times.

Syntax:
string str_repeat (string str, int number)

The function "repeats" a string *str number* times and returns the concatenated result.

```
echo str_repeat ("test!", 3); // outputs test! test! test!
```

str_pad Padding a
string with another string to a specified length.

Syntax :
string str_pad (string input, int pad_length [, string pad_string [, int pad_type]])

The *input* argument specifies the source string . The *pad_length* argument specifies the length of the returned string. If it is less than the original string, no addition is made. The optional *pad_string* argument can be used to specify which string to use as a placeholder (spaces by default). The optional argument *pad_type* can be used to specify *whether the* string should be padded to the right, left, or both. This argument can take the following values:

- STR_PAD_RIGHT (default)
- STR_PAD_LEFT
- STR_PAD_BOTH

```
$ str = "Aaaaa";
echo str_pad ($ str, 10);
// will return "Aaaaa"
echo str_pad ($ str, 10, "- =", STR_PAD_LEFT);
// will return "- = - = - Aaaaa"
echo str_pad ($ str, 10, "_", STR_PAD_BOTH)
// will return "_Aaaa_"
```

chunk_split
Returns a chunk of a string.

Syntax:
string chunk_split (string str [, int chunklen [, string end]])

The function **chunk_split ()** returns a string, wherein between each unit string *str* length *chunklen* sequence spacers inserted (default 76) *end* (default: "\ r \ n ").
This function can be useful when converting to "base64" format to comply with RFC 2045 rules.

```
// format $ data using RFC 2045 semantics
$ str = chunk_split (base64_encode ($ data));
```

This function is significantly faster than **ereg_replace ()** .

strtok
Returns the string in parts.

Syntax:
string strtok (string arg1, string arg2) The

function returns the part of the string *arg1* before the delimiter *arg2* . Subsequent calls return the next part up to the next delimiter, and so on until the end of the line. On the first call, the function takes two arguments: the original string *arg1* and the separator *arg2* . For each subsequent call , you do not need to specify the *arg1* argument , otherwise the first part of the string will be returned. When there is nothing more to return, the function will return false. If part of a string consists of 0 or an empty string, then the function will also return false.

```
$ str = "This is an example№string№ Aaa";
$ tok = strtok ($ str, "");
while ($ tok) {
  echo "$ tok";
  $ tok = strtok ("#");
};
// prints : "This" "is " "an" "example" "string"
```

It should be noted that the delimiters are a sequence of characters, each of which can individually be a delimiter, but when two or more delimiters are sequentially encountered in a string, the function returns an empty string (which can terminate the processing cycle, as in the example).

explode Splits a
string into an array.

Syntax:
array explode (string seperator, string str [, int limit])

The **explode ( )** function returns an array of strings, each of which corresponds to a portion of the original string *str* between the delimiters specified by *separator* .
The optional *limit* parameter specifies the maximum number of elements in the array. The remaining unseparated part will be contained in the last element.

```
$ str = "Path1 Path2 Path3 Path4";
$ str_exp = explode ("", $ str);
// now $ str_exp = array ([0] => Path1, [1] => Path2,
// [2] => Path3, [3] => '', [4] => Path4)
```

implode Concatenates an
array into a string.

Syntax:
string implode (string glue, array pieces)

The **implode ( )** function returns a string that contains sequentially all the elements of the array specified in the *pieces* parameter , between which the value specified in the *glue* parameter is inserted .

```
$ str = implode (":", $ arr);
```

join Joins an
array into a string.

Syntax :
: string the join (: string glue, pieces of array)

That same , that and **implode ()** .

Working with blocks of text

str_replace
Replaces one substring in the original string with another.

Syntax:
string str_replace (string from, string to, string str)

This function replaces all occurrences of the substring *from* (case sensitive) with *to* in *str* and returns the

result. The original string passed as the third parameter does not change. it can also work with binary strings.

substr_replace
Replaces one substring in the original string with another.

Syntax:
string substr_replace (string str, string replacement, int start [, int lemgth])

This function returns a string *str* in which the part from the character at position *start* and length *length* (or to the end if no length argument is specified) is replaced with the string *replacement* ...

If *start is* positive, the count is from the beginning of the string *str* , otherwise - from the end (-1 is the last character of the string).

If *length is* non-negative, then it indicates the length of the chunk to replace. If it is negative, then this is the number of characters from the end of the string *str* to the last character of the fragment being replaced (with a minus sign).

wordwrap
Splits the source text into lines with specific trailing characters.

Syntax:
string wordwrap (string str [, int width [, string break [, int cut]]])

This function splits a block of text *str* into multiple lines, terminated by *break* characters , so that there are no more than *width* letters on one line . The splitting occurs along the word boundary so that the text remains readable.

strtr
Complex replacement in a string.

Syntax:
string strtr (string str, string from, string to)
string strtr (string str, array from)

In the first case, the **strtr ()** function returns a string *str* , in which each character present in the string *from* is replaced with the corresponding string *to* ... If the strings *from* and *to are of* different lengths, then the extra trailing characters of the long string are ignored.

In the second case, the **strtr ()** function returns a string in which fragments of the *str* string are replaced with the values of the *from* array elements corresponding to the indices . In this case, the function tries to replace the largest fragments of the original string first and does not perform replacement in the already modified parts of the string. Thus, now we can perform several replacements at once:

```
$ Subs = array (
  "<name>" => "Larry",
  "<time>" => date ("dmY")
);
$ st = "Hello < name >! It's < time ";
echo strtr ($ st , $ Subs );
```

And here's how you can "undo" the action of the **HtmlSpecialChars ()** function :

```
$ Trans = array_flip (get_html_tra nslation_table ());
$ st = strtr ($ st , $ Trans );
```

As a result, we will get the original string from a string in which all special characters are replaced by their HTML equivalents.

stripslashes
Remove backslashes.

Syntax:
string stripslashes (string str);

Replaces some slash-preceded characters in *str* with their single-code equivalents. This applies to the following characters: ",", \.

stripcslashes
Convert special characters to their binary representation.

Syntax:
string stripcslashes (string str);

Returns a string in which those special characters that are commented out (for visual display) with a backslash are converted to their natural binary representation. C-like entries are recognized, for example: \ n, \ r ..., octal and hexadecimal sequences.

addslashes
Adds slashes before special characters in a string.

Syntax:
string addslashes (string str);

Inserts slashes only before the following characters: "," and \. It is very convenient to use this function when calling **eval ()** .

addcslashes
Formatting string with slashes in C-representation.

Syntax:
string addcslashes (string str, string charlist);

The **addcslashes ()** function returns the string *str* , into which the backslash "\" characters are inserted before the characters listed in the *ch arlist string* . This allows non-printable characters to be converted to their visual C representation.

quotemeta
Quoting metacharacters.

Syntax:
string quotemeta (string str);

Returns a string in which the added backslashes "\" in front of each of the following characters:
. \\ + *? [^] ($)

Can be used to prepare patterns in regular expressions.

strrev
Reverse a string.

Syntax:
string strrev (string str)

The **strrev ()** function returns the string *str* "backwards".

Functions for character conversion

nl2br
Replaces line feed characters.

Syntax:
string nl2br (string string)

Replaces all newline characters \ *n* in a string with <*br*> \ *n* and returns the result. The original string is not changed. Please note that the \ *r* characters that are present at the end of a line in Windows text files are not taken into account by this function, and therefore remain in the old place.

strip_tags
Removes tags from a string.

Syntax:
string strip_tags (string str [, string allowable_tags])

This function strips all HTML and PHP tags from the string and returns the result.
Incomplete or bogus tags cause an error.
The *allowable_tags* parameter can be used to pass tags that should not be removed from the string. They should be listed close to each other.

```
$ st = "
<b> Bold text </b>
<tt> Monospaced text </tt>
< a href = http : // spravkaweb . ru > Link </ a > ";
echo "Source: $ st";
echo "< hr > After removing tags:". StripTags ($ st , "< a > < b >").
    "<hr>";
```

By running this example, we can see that the <a> and <b> tags have not been removed (just like their paired closing ones), while the <tt> has disappeared.

get_meta_tags
The function searches and processes all <META> tags.

Syntax:
array get_meta_tags (string filename, int use_include_path)

The function opens the file and searches for all <META> tags in it until the closing </head> tag is encountered.
If the next <META> tag looks like:
<meta name = "title" content = "content">
then the pair *name => content is* added to the resulting array, which is returned at the end.
Special characters in the *filename* attribute value are replaced with an underscore "_", and alphabetic characters are converted to lowercase.
This function is convenient to use to quickly get all meta tags from a specified file.
If the optional *use_include_path* parameter *is* set, then the file is searched not only in the current directory, but also in all those specified for search by **include** and **require** statements .

get_html_translation_table
The function returns a translation table that is used by the htmlspecialchars () and htmlentities () functions.

Syntax:
string get_html_translation_table (int table [, int quote_style])

In this function, the *table* argument specifies which translation table to get: HTML_SPECIALCHARS for **htmlspecialchars ()** or HTML_ENTITIES for **htmlentities ()** . The optional *quote_style* parameter *is* described in the **htmlspecialchars ()** function .

```
$ trans = get _ html _ translation _ table ( HTML _ ENTITIES );
$ str = "<A & B>";
$ encoded = strtr ($ str, $ strans);
// $ encoded = "& amplt; A & amp; B & gt; "
```

Sometimes it is convenient to use the **array_flip ()** function to change the direction of transliteration.

```
$ trans = array_flip ($ trans);
$ original = strtr ($ encoded, $ trans);
```

htmlspecialchars
Converts special characters to HTML representation.

Syntax:
string HtmlSpecialChars (string str [, int quote_style]);

The main purpose of this function is to ensure that no part of the output line is interpreted as a tag. Replaces some characters in a string (such as the ampersand, quotation marks, and greater than and less than signs) with their HTML equivalents, so that they look like they are on the page. The most typical use of this function is the formation of the *value* parameter in various form elements so that there are no problems with quotation marks, or displaying a message in the guestbook if the user is not allowed to insert tags. You can specify what to do with quotes with the optional *quote_style* attribute :

- ENT_COMPAT (default) - only allow translation of double quotes
- ENT_QUOTES - enable translation of any quotes
- ENT_NOQUOTES - prohibit translation of any quotes

```
$ str = htmlspecialchars ("<a href=index.php> Home </a>", ENT_QUOTES);
```

htmlentities
Performs conversion of characters with HTML representation.

Syntax:
string htmlentities (string str [, int quote_style]);

This function is similar to **htmlspecialchars ()** , but only it does not perform selective translation, but full translation - for all characters that can have equivalent HTML representations.
You can specify what to do with quotes with the optional *quote_sty le* attribute :

- ENT_COMPAT (default) - only allow translation of double quotes
- ENT_QUOTES - enable translation of any quotes
- ENT_NOQUOTES - prohibit translation of any quotes

hebrev
Convert Boolean Hebrew text to displayable text.

Syntax:
string hebrev (string hebrew_text [, int max_chars_per_line]);

The optional *max_chars_per_line* argument specifies the number of characters per line of output. The function tries to avoid a word break.

hebrevc
Analogue of the hebrev function with hyphenation.

Syntax:
string hebrevc (string hebrew_text [, int max_chars_per_line]);

The **hebrevc ()** function is similar to **hebrev ()** with the difference that it converts the **newline** characters "\ n" to "
 \ n". the optional *max_cha rs_per_line* argument specifies the number of characters per line of output. The function tries to avoid word breaks.

quoted_printable_decode
Converts the quoted string to 8-bit.

Syntax :

string quoted _ printable _ decode (string str);

Case change functions

s trtolower
Converts string characters to lowercase.

Syntax:
string strtolower (string str);

Converts a string to lowercase. Returns the translation result.
It should be noted that if the locale is incorrectly configured, the function will produce, to put it mildly, strange results when working with Cyrillic letters.

```
$ str = "HeLLo World";
$ str = strtolower ($ str);
echo $ str;
// prints hello world
```

strtoupper
Converts the specified string to uppercase.

Syntax:
string strtoupper (string str);

Converts a string to uppercase. Returns the result of the transformation. This function works well with English letters, but it can be weird with Russian letters.

```
$ str = "Hello World";
$ str = strtoupper ($ str);
echo $ str;
// prints HELLO WORLD
```

ucfirst
Converts the first character of a string to uppercase.

Syntax:
string ucfirst (string str);

Returns a string whose first character is uppercase.
Cyrillic characters may not be converted correctly.

```
$ str = " hello world ";
$ str = ucfi rst ($ str );
echo $ str;
// prints Hello world
```

ucwords
Converts the first character of each word in a string to uppercase.

Syntax:
string ucwords (string str);

Returns a string that capitalizes the first character of each word in the string.
A word here means a section of a line preceded by a whitespace character: space, line feed, page feed, carriage return, horizontal and vertical tabs.
Cyrillic characters may not be converted correctly.

```
$ str = " hello world ";
$ str = ucfirst ($ str );
echo $ str;
// prints Hello World
```

Setting the locale (local settings)

setlocale
Sets regional settings.

Syntax:
string SetLocale (string category, string locale);

The **setlocale** function sets the current locale for case conversion, datetime display, and so on. Generally speaking, the locale is defined separately for each category of functions and looks different. Which category of functions will be affected by the call to **set locale ()** is specified in the *category* parameter . It can take the following string values:

- **LC_CTYPE** - activates the specified locale for upper / lower case translation functions;
- **LC_NUMERIC** - activates the locale for the formatting functions of fractional numbers - namely, specifies the separator of the integer and fractional parts in numbers;
- **LC_TIME** - sets the default date and time output format;
- **LC_ALL** - sets all of the above modes. Now let's talk about the *locale* parameter . As you know, each locale installed in the system has its own unique name by which it can be accessed. It is this that is fixed in this parameter. However, there are two important exceptions to this rule. Firstly, if the value of *locale* is the empty string "", it is set that the locale that is specified in the global environment variable with the same name as the category name *category* (or LANG - it is almost always present in Unix). Second, if 0 is passed in this parameter, then the new locale is not set, but the name of the current locale for the specified mode is simply returned.

```
setlocale ("LC_CTYPE", "ru_SU.KOI * -R");
// Here the call sets up the replacement table
// case of letters in accordance with the KOI8-R encoding.
```

Converting encodings

convert_cyr_string
Converts a string from one Cyrillic encoding to another.

Syntax :
string convert_cyr_string (string str, string from, string to);

The function converts the string *str* from encoding *from* in the encoding *to* . Of course, this makes sense only for strings containing "Russian" letters, since Latin in all encodings looks the same. Of course, the *from* encoding must match the true encoding of the string, otherwise the result will be incorrect. The *from* and *to* values are a single character specifying the encoding:

- k - koi 8- r
- w - wind ows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

The function is fast enough that it can be used to transcode letters into the desired form before sending them by email.

bin2hex
Converts character data to hexadecimal.

Syntax:

string bin2hex (string str)

The **bin2hex (**) function returns a string hexadecimal representation of the character-byte data contained in string *str* . Conversion is performed byte by byte, the most significant nibble is indicated first.

Format conversion functions

parse_url
Parses the URL and returns its components.

Syntax:
array parse_url (string url);

This function returns an associative array that includes many different existing URL components. These include "scheme", "host", "port", "user", "pass", "path", "query" and "fragment".
parse_str
Pushes URL strings into variables.

Syntax:
void parse_str (string str [, array arr]);

The **parse_str ()** function interprets the string *str* as if this string contained variables and their values and would be passed in the URL. Our function sets values for these variables.
If the second optional parameter is specified, then the values found using the **parse_str ()** function **are** stored not in global variables, but in the elements of the specified array.

```
$ str = "name [] = Vasia & name [] = Pupkin&id=12645&mail=vasia@mail.ru&url=www.vasia.ru";
parse_str ($ str);
parse_str ($ str, $ arr);
echo $ id; // prints 1264
echo $ name [0]; // will display Vasia
e cho $ name [1]; // prints Pupkin
print_r ($ arr);
// will output
Array
(
   [name] => Array
      (
         [0] => Vasia
         [1] => Pupkin
      )
   [id] => 12645
   [mail] => vasia@mail.ru
   [url] => www.vasia.ru
)
```

rawurlencode
URL encoding.

Syntax:
string RawUrlEncode (string str);

The **RawUrlEncode ()** function returns a string in which all non-alphanumeric characters (except for the hyphen "-", underscore "_", and period ".") Have been replaced by the sequences: percent sign (%) followed by two hexadecimal digits, which denote the character code. This encoding is necessary to ensure that alphabetic characters are not processed as URL string delimiters and are not distorted when transmitted over networks.

```
echo "<A href = ftp: // user:" .rawurlencode ($ mypasswd).
```

```
    "@ ftp.my.com / x.txt>"; // pass the password in the hyperlink
```

rawurldecode
Performs URL decoding.

Syntax:
string rawurldecode (string str);

This function returns a string that converts sequences with a percent sign (%) followed by two hexadecimal digits to the characters corresponding to this code. Similar to **urldecode ()** , but does not treat + as a space.

```
$ str = " foo % 20 bar % 40 baz ";
echo rawurldecode ($ str);
// prints foo bar @ baz
```

base64_encode Encodes the
data in MIME base64 encoding.

Syntax:
string base64_encode (string data);

base64_encode () returns base64 encoded data. This encoding is designed to transfer binary data through transport layers that do not contain the eighth bit, such as post bodies. Base64 encoded data takes up about 33% more space than the original.

base64_decode
Decodes data encoded in MIME base64 encoding.

Syntax:
string base64_decode (string encoded_data);

base64_decode () decodes encoded_data and returns the original data. The returned data can be binary.

URL functions

number_format
Formatting a number.

Syntax:
number_format ($ number, $ decimals, $ dec_point = ".", $ Thousands_sep = ",");

This function formats a floating point number by dividing it into triads with the specified precision. It can be called with two or four arguments, but not three! The *$ decimals parameter* specifies how many decimal places the number should have in the output string. The *$ dec_point parameter* is the integer and fractional separator, and the *$ thousands_sep* parameter is the triad separator in the number (if you specify an empty string in its place, the triads are not separated from each other).

Working with binary data

pack
Packing data into a binary string.

Syntax:
string pack (string format [, mixed $ args, ...]);

The **pack ()** function packs the given arguments into a binary string, which is then returned. The format of

the parameters, as well as their number, is specified using the *$ format* string , which is a set of one-letter formatting specifiers similar to those specified in **sprintf ()** , but without the% sign. Each specifier can be followed by a number that indicates how much information will be processed by this specifier. Namely, for the formats a, A, h and H, the number specifies how many characters will be placed in a binary string from those that are in the next string parameter when calling the function (that is, it determines the size of the field for outputting the string). In the case of @, it defines the absolute position at which the following data will be placed. For all other specifiers, the following numbers specify the number of arguments that this format applies to. Instead of a number, you can specify *, in this case, it is assumed that the specifier operates on all remaining data.

Here is a complete list of format specifiers:

- a - string, empty spaces in the field are filled with a character with code 0;
- A - string, empty spaces are filled with spaces;
- h - hexadecimal string, least significant bits at the beginning;
- H - hexadecimal string, high order bits at the beginning;
- c - signed byte (symbol);
- C - unsigned byte;
- s - signed short integer (16 bit byte order is determined by the processor architecture);
- S - unsigned short number;
- n - unsigned integer (16 bits, most significant bits at the end);
- v - unsigned integer (16 bits, least significant bits at the end);
- i is a signed integer (the size and byte order is determined by the architecture);
- I - unsigned integer;
- l - signed long integer (32 bits, the order of the characters is determined by the architecture);
- L - unsigned long integer;
- N - unsigned long integer (32 bits, most significant bits at the end);
- V - unsigned integer (32 bits, least significant bits at the end);
- f is a floating point number (depends on the architecture);
- d - double precision floating point number (depending on architecture);
- x - a character with a zero code;
- X - go back 1 byte;
- @ - filling with a zero code to the specified absolute position.

```
// Whole, whole, everything else is sivol
$ bindata = pack (" nvc *", 0 x 1234, 0 x 5678, 65, 66);
```

After executing the above code, the *$ bindata* line will contain 6 bytes in the following sequence: 0x12, 0x34, 0x78, 0x56, 0x41, 0x42 (in hexadecimal notation).

unpack
Unpacks data from a binary string.

Syntax:
array unpack (string format, string data);

Unpacks data from a binary string into an array according to the format. Returns an array containing the unpacked elements.

```
$ array = unpack (" c 2 chars / nint ", $ binarydata );
```

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

String Sums and Hash Functions

strlen
Returns the length of the string.

Syntax:
int strlen (string str)

Returns simply the length of the string, i.e., how many characters *str* contains .
The string can contain any characters, including those with a zero code. The **strlen ()** function will work

correctly with such strings as well.
count_chars
Returns information about the characters in a string.

Syntax:
mixed count_chars (string str [, int mode])

The **count_chars (**) function counts the frequency of each byte (0-255) in *str* and returns the result in an array according to the optional *mode* argument . *mode* can take on the following values:
- 0 (default) - an array with bytes as indices and repetition rate as values of an array element
- 1 - similar to 0, but bytes missing in *str are* not returned
- 2 - similar to 0, but only those bytes that are missing are returned
- 3 - returns a string consisting of all found characters
- 4 - returns a string consisting of all missing characters

md5
Get the MD5 hash string.

Syntax:
string md5 (string str);

Returns the hash code of *str* , based on the RSA Data Security Corporation's MD5 Message-Digest Algorithm. The hashcode is just a string, practically unique to each of the *str* lines . That is, the probability that two different strings passed to *str* will give us the same hash code tends to zero.
At the same time, if the length of a string *str* can reach several thousand characters, then its MD5 code takes a maximum of 32 characters.

crc32
Get the crc32 string polyminal.

Syntax:
int crc32 (string str); **Crc32 ()**

function calculates the 32-bit checksum of *str* . That is, the result of its operation is a 32-bit (4-byte) integer. Usually this function is used to check the integrity of the transmitted data. This function is much faster than **md5 ()** , but at the same time produces much less reliable "hash codes" for a string. So, now, to get the same "hash codes" by random selection for two different strings, you will need not a trillion years of operation of the most powerful computer, but only a year or two.

crypt
Performs symmetric encryption.

Syntax:
string crypt (string str [, string salt]);

The *str* argument specifies the string to be encrypted.
The hash code for the same string, but with different *salt* values (by the way, this must be a two-character string) gives different results. If the *salt* parameter is omitted, PHP will randomly generate it.
On systems that support multiple encryption algorithms, the following constants are set to 1 or 0, depending on whether the algorithm is supported or not:
- CRYPT_STD_DES - standard 2-byte DES encryption (SALT = 2)
- CRYPT_EXT_DES - extended 9-byte DES encryption (SALT = 9)
- CRYPT_MD5 - 12-byte MD5 encryption (SALT starts at $ 1 $)
- CRYPT_BLOWFISH - extended 12-byte DES-encryption (SALT starts with $ 2 $)

. Since this function uses a one-way encryption algorithm, there is no decryption function.

metaphone Computes a

metaphone hash.

Syntax:
string metaphone (string str);

This function is similar in action to **soundex ()** , it calculates the pronunciation code of the word passed in the string *str* , but with increased calculation accuracy, since uses English pronunciation rules.
The returned string value can be of variable length.

soundex
Calculate the pronunciation similarity hash.

Syntax:
string soundex (string str);

The **soundex ()** function is used to check spelling when you know approximately how a word sounds, but you don't know how to spell it, and you have a dictionary (database) against which you can check.
A 4-character string is returned: the first letter of the word and 3 numbers.

```
soundex (" Euler ") == soundex (" Ellery ") == " E 460";
soundex ("Gauss") == soundex ("Ghosh") == "G200";
soundex ("Hilbert") == sou ndex (" Heilbronn ") == "H416";
soundex ("Knuth") == soundex ("Kant") == "K530";
soundex ("Lloyd") == soundex ("Ladd") == "L300";
soundex ("Lukasiewicz") == soundex ("Lissajous") == "L222";
```

Symbolic links. Hard links

A bit of theory

On Unix systems, it is quite common to have different names for the same file or directory. In this case, it is logical to call one of the names the main one, and all the others - his pseudonyms. In Unix terminology, such aliases are called *symbolic links* .

A symbolic link is simply a special kind of binary file that contains a link to the main file. When accessing such a file (for example, opening it for reading), the system "understands" which object is actually being accessed and provides it transparently. This means that we can use symbolic links just like regular files. However, sometimes you need to work with a link exactly as a link, and not as a file. This is what the PHP functions listed below are for.

Hard links

Creating a symbolic link is not the only way to give multiple names to the same file. The main drawback of symbolic links is the existence of a base filename that everyone links to. Try to delete this file and the whole web of links, if any, will fall to pieces. There is another drawback: opening the file to which the link points is somewhat slower, because the system needs to parse the contents of the link and establish a link to the "real" file. This is especially felt if one link points to another, and that one to the third, etc. levels by 10.

Hard links allow you to have several completely equal names for one file, and access to them is carried out equally quickly. Moreover, if one of these names is deleted, then the file itself will be deleted only if this name was the last one, and the file has no other names.

You can register a new file name (that is, create a hard link for it) using the **link ()** function . Its syntax is completely identical to the **symlink ()** function , and it works according to the same rules, except that it creates not a symbolic link, but a hard link.

readlink
Returns the name of the main file.

Syntax:
string readlink (string $ linkname)

Returns the name of the main file with which its synonym *$ linkname is associated* . This is useful if you want to know the base name of a file so that, for example, you can delete the file itself, rather than a link to it. On error, the function returns false.

symlink
Creates a symbolic link.

Syntax:
bool symlink (string $ target, string $ link)

This function creates a symbolic link named *$ link* to the object (file or directory) specified in *$ target* . If it fails, the function returns false.

lstat This
function gathers together all the information provided by the operating system for the specified link and returns it as an array.

Syntax:
array lstat (string $ filename)

The function is exactly the same as calling **stat ()** , except that if *$ filename* specifies not a file, but a symbolic link, information about this link will be returned (and not about the file it points to as **stat ()** does).

linkinfo
The function returns the value of the "device" field from the result returned by the **lstat ()** function .

Syntax:
int linkinfo (string $ linkname)

This is usually used when you want to determine if the object pointed to by a symbolic link in *$ linkname still exists* .

Date and time functions

checkdate
Checks if the date / time is correct.

Syntax:
int checkdate (int month, int day, int year);

The **checkdate ()** function checks the validity of the date given in its arguments.
Returns true if the date specified as "month, day, year" (month, day, year) is correct, otherwise false. A date is considered correct if:
- a year between 1 and 32767 inclusive
- a month between 1 and 12 inclusive
- the day is in the range of allowed days for this month. Leap years are counted .

```
$ month = 1;
$ day = 10;
$ year = 2002;
if (checkdate ($ month, $ day, $ year)) echo " This day there !";
else echo "There is no such day!";
```

Output: There is such a day!

```
$ month = 13;
$ day = 10;
$ year = 2002;
if (checkdate ($ month, $ day, $ year)) echo " This day there !";
else echo "There is no such day!";
```

Output: There is no such day!

date
Local time / **date** format.

Syntax:
string date (string format [, int timestamp]);

This function returns a string containing the date and time formatted according to the *format* string and using the *timestamp* or the current local time if no timestamp is specified.
The following characters must be used in the format string:

- a - "before" and "after" noon: "am" or "pm"
- A - "Before" and "Afternoon": "AM" or "PM"
- d - day of the month, 2 digits (zero in the first place) (from 01 to 31)
- D - day of the week, text, 3 letters; those. "Fri"
- j - day of the month, 1-2 digits without leading zeros (from 1 to 31)
- F - month, text, long; those. "January"
- h - hour, 12-hour format (from 01 to 12)
- H - hour, 24-hour format (from 00 to 23)
- g - hour, 12-hour format without zeros (from 1 to 12)
- G - hour, 24-hour format without zeros (from 0 to 23)
- i - minutes (from 00 to 59)
- I (large i) - 1 if daylight saving time is in effect, otherwise 0
- L - 0 if the year is not a leap year, or 1 otherwise
- B - Swatch Internet time
- T - computer time zone, for example: MDT (not always available)
- l (lowercase "L") - day of the week, text, long; those. "Friday"
- m - month, two digits followed by zeros (from 01 to 12)
- n - month, one or two digits without zeros (from 1 to 12)
- M - three-letter English abbreviation of the month; those. "Jan"
- t - number of days in the specified month (from 28 to 31)
- s - seconds (from 0 to 59)
- S - English-language ordinal suffix of a number of two letters, text, ie. "th", "nd"
- U is an integer number of seconds since the beginning of the UNIX era (not always available)
- Y - year, numeric, 4 digits (1999)
- y - year, numeric, 2 digits (99)
- w - ordinal number of the day in the week, (from 0-Sunday to 6-Saturday)
- z - ordinal number of a day in a year (from 0 to 365)
- Z - mixing time zone in seconds (from -43200 to 43200)

All other characters in the format string argument are returned as is in the resulting string.
The "Z" format always returns 0 when used with the gmdate () function.

```
echo date ("Today is d . m . Y ");
// Today is 01/31/2002
echo date ("l dS o f FY h: i: s A");
// Thursday 31st of January 2002 12:51:19 PM
echo " July 1, 2000 is on a". date ("l", mktime (0,0,0,7,1,2000));
// July 1, 2000 is on a Saturday
```

The date () and mktime () functions can be used together to find dates in the future or the past.

```
$ tomorrow = mktime (0,0,0, date ("m"), date ("d") + 1, date ("Y"));
$ lastmonth = mktime (0,0,0, date ("m") - 1, date ("d"), date ("Y"));
$ nextyear = mktime (0,0,0, date ("m"), date ("d"), date ("Y") + 1);
```

localtime
Gets date / time information.

Syntax:
array localtime ([int timestamp [, bool is_associative]]);

The first optional argument to this function is a Unix timestamp. If it is not specified, the current time is used.

If the second optional parameter is zero (the default), then the returned array will be numerically indexed; otherwise, an associative array is returned where the elements have the following meanings:

- ([1]) "tm_sec" - seconds
- ([2]) "tm_min" - minutes
- ([3]) "tm_hours" - hours
- ([4]) "tm_mday" - day of the month
- ([5]) "tm_mon" - month of the year
- ([6]) "tm_year" - year, digital
- ([7]) "tm_wday" - day of the week
- ([8]) "tm_yday" - day of the year
- ([9]) "tm_isdst" - whether daylight saving time is active

gettimeofday
Get the date with a system call.

Syntax:
array gettimeofday ();

This function returns an associative array that contains the date returned by the system call. The function is the interface of the gettimeofday (2) system function.
The returned associative array contains the following elements:

- "sec" - seconds
- "usec" - microseconds
- "minuteswest" - offset west of Greenwich, in minutes
- "dsttime" - type of dst correction (daylight saving time)

strftime
Format time according to local settings.

Syntax:
string strftime (string format [, int timestamp]);

Returns a string formatted according to the given format string *format* and using the given *timestamp* or the current local time if no stamp is specified.
Using the **setlocale ()** function, you can set the language in which the names of months and days will be displayed.
The following conversion specifiers should be used in the format string:

- % a - abbreviated name of the day of the week by default (Wed);
- % A - full name of the day of the week by default (Wednesday);
- % b - abbreviated month name by default (Apr);
- % B - full month name by default (April);
- % c is the preferred date and time representation (06/19/02 15:45:11);
- % C - century number (year divided by 100 and without fractional part, from 00 to 99);
- % d - day of the month as a decimal number (in the range from 0 to 31);
- % D - equivalent to% m /% d /% y;
- % e - day of the month (a space is inserted instead of a non-significant zero) (from 1 to 31);
- % h - analogue of% b;
- % H - hour as a decimal number in 24-hour format (in the range from 00 to 23);
- % I - hour as a decimal number in 12-hour format (in the range from 01 to 12);
- % j - number of the day in the year as a decimal number (in the range from 001 to 366);
- % m - month number as a decimal number (in the range from 1 to 12);
- % M - minutes as a decimal number;
- % n - newline character;
- % p - "am" or "pm" (before and after noon) according to the current time;
- % r - time in 12-hour format (am or pm);
- % R - time in 24-hour format;
- % S - seconds as a decimal number;
- % t is a tab character;
- % T - current time, equivalent to% H:% M: $ S;

- % u - number of days in the week (from 1 to 7) (Monday - 1);
- % U is the number of the week of the year as a decimal number, starting with the first Sunday as the first day of the first week;
- % V is the number of the week of the year according to the ISO 8601: 1988 standard (from 1 to 53), where the first week is the one with more than 3 days in the current year;
- % W - week number of the year as a decimal number, starting with the first Monday as the first day of the first week;
- % w - the number of the day in the week (from 0 to 6) (Sunday - 0);
- % x - representation of the date in the system format without specifying the time (06/13/02);
- % X - representation of time in system format without specifying the date (15:34:54);
- % y - year as a decimal number without century (in the range from 00 to 99);
- % Y - year as a decimal number, including the century;
- % Z - time zone or name or abbreviation;
- %% - symbol "%".

```
setlocale ("LC_TIME", "C");
print (strftime ("% A in Finnish is") );
setlocale ("LC_TIME", "fi");
print (strftime ("% A, in French"));
setlocale ("LC_TIME", "fr");
print (strftime ("% A and in German"));
setlocale ("LC_TIME", "de");
print (strftime ("A. \ n"));
```

Formats local time according to locale setting.

getdate

Gets date / time information.

Syntax:

array getdate (int timestamp);

Returns an associative array containing date information with the following elements:

- " seconds " - seconds
- " minutes " - minutes
- "hours" - hours
- "mday" - day of the month
- "wday" - day of the week, numeric
- "mon" - month, numeric
- "year" - year, digital
- "yday" - day of the year, digital; those. "299"
- "weekday" - day of the week, text, full; those. "Friday"
- "month" - month, text, full; those. "January"
- 0 - "UNIX timestamp" argument received .

```
print_r (getdate (time ()));
```

The above example will output the following:

```
Array
(
    [seconds] => 23
    [minutes] => 44
    [hours] => 22
    [mday] => 15
    [wday] => 0
    [mon] => 8
    [year] => 2004
    [yday] => 227
    [weekday] => Sunday
    [month] => August
    [0] => 1092595463
)
```

gmdate

Get the date in a formatted string for GMT time.

Syntax:
string gmdate (string format, int timestamp);

Same as date () except that the time is returned in Greenwich Mean Time (GMT). For example, when running in Finland (GMT +0200), the first line below will print "Jan 01 1998 00:00:00", while the second line will print "Dec 31 1997 22:00:00".

```
echo date ("M d YH: i: s", mkti me (0,0,0,1,1,1998));
echo gmdate ("M d YH: i: s", mktime (0,0,0,1,1,1998));
```

gmstrftime
Local time / date formatting.

Syntax:
string gmstrftime (string format, int timestamp);

This function works in the same way as strftime (), except that it returns Greenwich Mean Time (GMT). For example, if you run in the zone (GMT -0500), the first line is "Dec 31 1998 20:00:00", and the second is "Jan 01 1999 01:00:00".

```
setlocale ("LC_TIME", "en_US");
echo strftime ("% b% d% Y% H:% M:% S", mktime ( 20,0,0,12,31,98)). "\ n";
echo gmstrftime ("% b% d% Y% H:% M:% S", mktime (20,0,0,12,31,98)). "\ n";
```

mktime
Gets the UNIX timestamp for a date.

Syntax :
int mktime ([int hour] [, int minute] [, int second] [, int month] [, int day] [, int year] [, int is_dst]);

Returns the temporary label Unix according to data arguments . This timestamp is an integer equal to the number of seconds between the Unix epoch (Jan 1, 1970) and the specified time. All parameters of these functions are optional, but they can only be skipped from right to left. If some parameters are not specified, values corresponding to the current date are substituted in their place. The *is_dst* argument indicates whether daylight saving time (1) has *occurred* or not (0); if not known, the argument is (-1) The function returns the *timestamp* corresponding to the specified date. The correctness of the date passed in the parameters is not checked. In the case of an incorrect date, nothing special happens - the function "pretends" that it does not concern it, and forms the corresponding *timestamp* .

```
echo date ("MdY", mktime (0,0,0,12,32,1997)); // correct date
echo date ("MdY", mktime (0,0,0,13,1,1997)); // wrong date
echo date ("MdY", mktime (0,0,0,1,1,1998)); // wrong date
```
Outputs three identical numbers

gmmktime
Analog of the time () function for GMT time.

Syntax:
int gmmktime (int hour, int minute, int second, int month, int day, int year [, int is_dst]);

Identical to mktime () except that the parameters passed are in Greenwich Mean Time (GMT).

time
Get time in seconds.

Syntax:
int time ();

Returns the current time, measured in seconds since the Unix epoch (Jan 1, 1970 00:00:00 GMT).
This data format is accepted in Unix as a standard (called "UNIX timestamp"): in particular, the time of the last modification of files is indicated in this format. Generally speaking, almost all functions for working from time to time deal with this representation of it (called the *timestamp*). That is, the representation "number of seconds since January 1, 1970" is very versatile and, what is most important, convenient.

```
echo time ();
```

microtime
Returns the current UNIX timestamp in microseconds.

Syntax:
string microtime ();

Returns the string "msec sec" where sec is the current time, measured in seconds since the Unix epoch (0:00:00 January 1, 1970 GMT), and msec is the fraction in microseconds. These functions are only available on operating systems that support the gettimeofday () system call.
But the point is that milliseconds look different in different OCs. For example, in Unix it is really a number of microseconds, but in Windows it is an incomprehensible value.

```
echo microtime (); // in Windows it will display something like 0.53033200 1012468870
```

strtotime
Lexical conversion of time string to Unix timestamp.

Syntax:
int strtotime (string time [, int now]);

In the *time* argument, the function receives a date in English format and then converts it to an integer Unix timestamp format.

```
echo strtotime ("now"). "\ n";
echo strtotime (" 10 September 2002 "). "\ n";
echo strtotime ("+ 2 day"). "\ n";
echo strtotime ("+ 3 week"). "\ n";
echo strtotime ( "+1 week 2 days 4 hours 34 seconds"). "\ n";
```

Logical functions for determining the type of a variable

is_scalar
Checks if a variable is simple.

Syntax:
bool is_scalar (mixed var)

Returns true if *var* is of scalar type (chill, strings, boolean values) but not complex type (arrays or objects).

is_null
Checks if a variable is empty.

Syntax:
bool is_null (mixed var)

Returns true if *var* does not exist (or has been explicitly assigned a null value). The function is equivalent to the expression:
(var === null or is_set (var)

is_numeric

Checks if a variable is numeric.

Syntax:
bool is_numeric (mixed var)

Returns true if *var* is of a numeric type (integer, double), or a string with a numeric value.

is_bool
Checks if a variable is boolean.

Syntax:
bool is_bool (mixed var)

Returns true if *var* is a boolean type (TRUE or FALSE), otherwise false.

is_int
Determines if a variable is of type integer.

Syntax:
bool is_int (mixed var);

Returns true if var is of type integer.

is_integer
Determines if a variable is of type integer.

Syntax:
bool is_integer (mixed var);

Returns true if var is of type integer, or false otherwise.

is_long
Determines if a variable is of type integer.

Syntax:
bool is_long (mixed var);

Returns true if var is of an integer type (integer, long), or false otherwise.

is_real
Determines if a variable is of type real (fractional).

Syntax:
bool is_real (mixed var);

Returns true if var is of type real (fractional), or false otherwise.

is_float
Determines if a variable is of type float (fractional).

Syntax:
bool is_float (mixed var);

Returns true if var is of type float (fractional), or false otherwise.
is_double
Determines whether a variable is of type double (fractional).

Syntax:
bool is_double (mixed var);

Returns true if var is of type double (fractional), or false otherwise.

is_string
Determines if a variable is a string.

Syntax:
bool is_string (mixed var);

Returns true if var is a string, or false otherwise.

is_array
Determines if a variable is an array.

Syntax:
bool is_array (mixed var);

Returns true if var is an array, or false otherwise.

is_object
Determines if a variable is an object.

Syntax:
bool is_object (mixed var);

Returns true if var is an object, or false otherwise.

is_resource
Determines if a variable is a resource pointer.

Syntax:
bool is_resource (mixed var);

Returns true if var points to a resource allocated and returned by the intended function.

Resources are objects like files or database query results that are allocated and released by PHP's internal functions. When a program no longer requires a resource, it is good programming practice to explicitly release it with the intended functions. But in most cases, the PHP interpreter frees up unnecessary resources as needed (usually at the end of the script).

get_resource_type
Determines the type of the resource descriptor.

Syntax:
string get_resource_type (resource $ handle);

This function returns a string containing a description of the resource type. If an invalid resource pointer is passed in the argument, an error occurs.

```
$ c = mysql_connect ();
echo get_resource_type ($ c). "\ n";
// prints : mysql link

$ file = fopen ( "filename.txt", "w");
echo get_resource_type ($ file). "\ n";
// will output : file

$ doc = new_xmldoc ("1.0");
echo get_resource_type ($ doc). "\ n";
// will output: domxml document
```

Variable functions

gettype
Gets the type of the variable.

Syntax:
string gettype (mixed var);
Returns the type of PHP variable var.
Possible values for the returned string:
"integer"
"double"
"string"
"array"
"object"
"unknown type"

intval
Returns the integer value of a variable.

Syntax:
int intval (mixed var, int [base]);

Returns the integer value of var using the specified translation base (default 10). var can be scalar. You cannot use the intval () function for arrays or objects.

doubleval
Gets the value of a variable in double format.

Syntax:
double doubleval (mixed var);

Returns the double (floating point) value of var.
var can be scalar. You cannot use doubleval () on arrays and objects.

empty
Determines if the variable has any value.

Syntax:
int empty (mixed var);

Returns false if var exists and is not null or null; true otherwise.
The function actually checks if the variable has a value that can be set to 0, that is: (var == 0)

```
$ var = 0;
if ( empty ($ var )) {
  echo "$ var is either 0 or not important";
  if (! isset ($ var )) {
    echo "$ var doesn't matter";
  };
};
```

Note that it is useless to use this function with an argument other than a variable, for example the expression empty (Addslashes ($ name)) is meaningless, since here the value returned by the function is checked. The only thing that the **empty ()** function can reveal in this variant is whether the expression has a value equivalent to true (not equal to zero), and this can be checked without using the function.

isset
Determines if a variable exists.

Syntax:
int isset (mixed var);

Returns true if var exists; false otherwise.
The function actually checks if the variable has a value identical to null, that is: (var === null). Note the difference between equality and identity.
If a variable was removed with unset (), it will no longer be determined by isset ().

```
$ a = "test";
echo isset ($ a); // true
unset ($ a);
echo isset ($ a); // false
```

settype
Sets the type of the variable.

Syntax:
int settype (string var, string type);

Sets the type of variable var to type.
Possible values of type:
"integer"
"double" "
" string "
" array ""
"object"
Returns true on success; false otherwise.

str val
Gets the string value of a variable.

Syntax:
string strval (mixed var);

Gets a var string value.
var can be of any scalar type. You cannot use strval () on arrays or objects.

unset
Removes the specified variable.

Syntax:
int unset (mixed var);

unset () destroys the specified variable and returns true.
Example :
unset ($ foo);
unset ($ bar ["quux"]);

Functions for working with functions

get_defined_functions
Returns a list of all available functions.

Syntax:
array get_defined_functions ()

Function **get_defined_functions ()** Returns a multidimensional array that contains the names of all functions available to the script.

```
print_r (get_defined_functions);
```

function_exists
Checks for the existence of a function.

Syntax:
bool function_exists (fuction_name: string)

function **function_exists ()** returns the value true, if the function named *function_name* there in the script. Returns false otherwise.

```
if ( function _ exists (" imagecreate ")) {
  echo "Graphics library available!
";
} else {
  echo "Graphics library not available!
";
};
```

call_user_func
Performs an indirect function selection.

Syntax :
mixed call_user_func (string function_name [, mixed parameter [, mixed ...]])

The **call_user_func ()** function calls the *fu nction_name* function and passes all the other parameter *parameters ...*

```
<? php
function myFunc ($ str) {
  echo $ str;
};
call_user_func ("myFunc", "Hello World");
?>
```

create_function
Dynamically create a function.

Syntax:
string create_function (string args, string code)

The **create_function ()** function creates an anonymous function and returns the name generated for that function. Function arguments listed in the *args* argument *are* usually quoted in single quotes. The body of the function is also passed in the *code* argument . This is necessary to prevent the interpreter from replacing variables with values. If you still limit it with double quotes, then you must precede the variable with a slash: \ $ var.

Usually, names returned by a function are prefixed with *lambda_* .

Using this function, you can create functions based on information obtained during script execution.

```
$ func = create_function ('$ a, $ b',
         'return "$ a * $ b =". $ a * $ b);');
echo "New function name: $ func <br>";
echo $ func (2,3);
```

This example will output the following:

```
New function name: lambda_1
6
```

func_get_arg
Get the function argument.

Syntax:
mixed func_get_arg (int arg_num);

The **func_get_arg ()** function returns the specified *arg_num* agrumen that was passed to the current user-defined function as a parameter. The parameters passed to the function start from zero.

If this function is called outside the function definition, then it issues an error warning. Also, a warning will be issued when trying to find out a parameter that does not exist in the list of arguments (the function will return false). In order for the function to work correctly, it is necessary to know in advance the total number of parameters that are passed to the user-defined function using the **func_num_args ()** function .

```
<? php
functi on func () {
  $ num_args = func_num_args ();
  echo "The number of arguments for this function is: $ num_args <br>";
  for ($ i = 0; $ i <$ num_args; $ i ++)
    echo "$ i argument :" .func_get_arg ($ i). "<br>";
};

func ("2", 1, "tree");
?>
```

func_get_args
Get function arguments in an array.

Syntax:
array func_get_args ();

The **func_get_args ()** function returns the list of arguments in an array with which the current user-defined function was called. If **func_get_args ()** is called outside the definition of a user-defined function, an error warning is issued.

```
<? php
function func () {
  $ num_args = func_num_args ();
  echo "The number of arguments for this function is: $ num_args <br>";
  $ func_list = func_get_args ();
  for ($ i = 0; $ i <$ num_args; $ i ++)
    echo "$ i argument number $ i:". $ func_list [$ i]. "<br>" ;
};

func ("2", 1, "tree");
?>
```

func_num_args
Returns the number of arguments received in a custom function.

Syntax:
int func_num_args ();

Function **func_num_args ()** returns the number of arguments that were passed into the current user-defined function.

Typically this function is used in conjunction with **func_get_arg ()** and **func_get_args ()** in custom functions that can take an indefinite number of parameters.

```
<? php
function func () {
  $ num_args = func_num_args ();
  echo "The number of arguments for this function is: $ num_args <br>";
  $ func_list = func_get_args ();
  for ($ i = 0; $ i <$ num_args; $ i ++)
    echo "$ i argument number $ i:". $ func_list [$ i]. "<br>";
};

func ("2", 1, "tree");
?>
```

Calendar functions

JDToGregorian
Converts daily Julian account to Gregorian date.

Syntax:
string jdtogregorian (int julianday);

Converting daily Julian account to Gregorian in month / day / year format
GregorianToJD
Converts a Gregorian date to a Julian Day Count.

Syntax:
int gregoriantojd (int month, int day, int year);

The correct range for the Gregorian calendar is 4714 CE. before 9999 AD

While this software can reverse dates back to A.D. 4714, such use may be useless and not significant. The Gregorian calendar was not instituted until October 15, 1582 (or October 5, 1582 in the Julian calendar). Some countries did not accept it for a very long time. For example, Great Britain was transformed in 1752, the USSR in 1918 and Greece in 1923. Most European countries used the Julian Calendar before the Gregorian.
Example:
<? php
$ jd = GregorianToJD (10,11,1970);
echo ("$ jd \ n");
$ gregorian = JDToGregorian ($ jd);
echo ("$ gregorian \ n");
?>

JDToJulian
Converts a Julian date to a Julian day count.

Syntax:
string jdtojulian (int julianday);

Converts a daily Julian account to a string containing a Julian calendar date in month / day / year format.

JulianToJD
Converts a Julian calendar date to a daily Julian count.

Syntax:
int juliantojd (int month, int day, int year);

The correct range for the Julian calendar is AD 4713. before 9999 AD

Although this software can operate in reverse order up to AD 4713. such use may be useless and not significant. The calendar was created in 46 AD, but the detailed did not stabilize until at least 8 AD, and possibly later in the 4th century. Also, the beginning of the year is different from one culture to another - not everyone agrees that January is the first month.

JDToJewish
Converts the daily Julian count to the Hebrew calendar.

Syntax :
string jdtojewish (int julianday);

JewishToJD
Converts a date in the Hebrew Calendar to a daily Julian count.

Syntax:
int jewishtojd (int month, int day, int year);

The Hebrew calendar has been in use for several millennia, but during the initial period there was no formula to determine the beginning of the month. The new month began when the full moon was seen.

JDToFrench
Converts Daily Julian Account to French Republican Calendar.

Syntax:
string jdtofrench (int month, int day, int year);

Converts Daily Julian Account to French Republican Calendar.

FrenchToJD
Converts the date and the French Republican calendar to the Julian day count.

Syntax:
int frenchtojd (int month, int day, int year);

This program converts dates from 1 to 14 (Gregorian dates September 22, 1792 to September 22, 1806). This covers the period when the calendar was used.

JDMonthName
Returns the name of the month.

Syntax:
string jdmonthname (int julianday, int mode);

Returns a string with the name of the month. main tells the function which calendar to convert the Julian day account to, and what type of monthly names should be returned.
Calendar ways

0	Gregorian - apreviated
1	Gregorian
2	Julian - apreviated
3	Julian
4	Jewish
five	French Republican

JDDayOfWeek
Returns the day of the week.

Syntax:
mixed jddayofweek (int julianday, int mode);

Returns the day of the week. It can return a string or an int depending on the mode.
Calendar weekly paths

Way	Value
0	returns the day number as int (0 = Sunday, 1 = Monday, etc.)
1	return string content of the day of the week (english-gregorian)
2	returns a string containing abreviated days of the week (English-Gregorian)

1

Part 3. Files and networks

Working with files

Opening a file

fopen
Opens a file and binds it to a descriptor.

Syntax:
int fopen (string $ filename, string $ mode, bool $ use_include_path = false)

Opens a file named *$ filename* in *$ mode* and returns a handle to the open file. If the operation failed, then the function returns false. The optional parameter *use_include_pa th* tells that if a relative file name is given, it should also be looked for in the list of paths used by the **include** and **require** statements . This parameter is usually not used.
The *$ mode* parameter can take the following values:
r
- The file is opened for reading only. If the file does not exist, the call logs an error. After successful opening, the file pointer is set to its first byte, i.e. at the beginning.
r +
- The file is opened simultaneously for reading and writing. The current position pointer is set to its first byte. Returns false if the file does not exist. If at the time of recording the file pointer is set somewhere in the middle of the file, then the data will be written directly over the existing ones, and not spread them apart, increasing the file size if necessary.
w
- Creates a new empty file. If at the time of the call there was already a file with the same name, then it is previously destroyed. In the case of an incorrectly specified file name, the call fails.
w +
- Similar to r +, but if the file did not exist initially, creates it. After that, you can work with the file in both read and write modes. If the file existed before the call, its contents are deleted.
a
- Opens an existing file in write mode, and at the same time shifts the current position pointer by the last byte of the file. As usual, the call fails if the file is missing.
a +
- Opens the file in read / write mode, the file pointer is set to the end of the file, while the contents of the file are not destroyed. It differs from a in that if the file did not originally exist, then it is created. This mode is useful if you need to add something to a file, but you don’t know if such a file has already been created.

But this is not yet a complete description of the *$ mode* parameter . The fact is that at the end of any of the strings r, w, a, r +, w + and a + there can be one more optional character - b or t. If b is specified (or none at all), then the file is opened in binary read / write mode. If this is t, then the translation mode of the line feed character is set for the file, i.e. it is perceived as textual.

tmpfile
Creates a new temporary file with a unique name and opens it for reading and writing.

Syntax:
int tmpfile ()

Further work must be done with the returned file descriptor, because the file name is not available.

The space occupied by a temporary file is automatically released when it is closed and when the program exits.

Closing the file

fclose
Closes a file previously opened with **fopen ()** .

Syntax:
int fclose (int $ fp)

Returns false if the file could not be closed (for example, something happened to it or the connection with the remote host was broken). Otherwise, it returns true.

You should always close FTP and HTTP connections, because otherwise a "stray" file will lead to unnecessary downtime and unnecessary server load. In addition, by successfully closing the connection, you can be sure that all data was delivered without errors.

Read and write

fread
Reads a specified number of characters from an open file.

Syntax:
string fread (int $ f, int $ numbytes)

Reads *$ f $ numbytes* characters from the file and returns a string of those characters. After reading, the file pointer advances to the next position after the read block. If *there is* more *$ numbytes* than can be read from the file, what was read is returned. This technique can be used if you need to read the entire file into a line. To do this, just set *$ numbytes to a* very large number. But if you are concerned about saving memory on the system, this is not recommended.

fwrite
Write to file.

Syntax:
int fwrite (int $ f, string $ str)

Writes the entire contents of string *$ str* to file *$ f* . This function pairs with **fread ()** in reverse. When working with text files (that is, when the t character is specified in the file open mode), all \ n are automatically converted to the line separator accepted in your operating system.

fgets
Reads a single line from the file, ending with a newline character \ n.

Syntax:
string fgets (int $ f, int $ length)

This character is also read and included in the result. If a line in the file is more than *$ length-1* bytes, then only its *$ length-1* characters are returned . This function is useful if you have opened a file and want to loop through all of its lines. However, even in this case (and faster), the **File ()** function will be used . It is also worth noting that this function (like the **fread ()** function) in the case of text mode on Windows takes care of converting \ r \ n pairs to a single \ n character.

fputs
Complete analogue of **fwrite ()** . **Syntax :** int fputs (int $ f, string $ str)

fgetcsv
Function for working with one of the file formats in which Excel data can be saved.

Syntax:
list fgetcsv (int $ f, int $ length, char $ delim = ",")

The function reads a line from the file specified by descriptor *$ f* and splits it at the *$ delim* symbol . The *$ delim parameter* must be a single character string, otherwise only the first character of this string is taken into account. The function returns the resulting list, or false if the lines ran out. The *$ length* parameter specifies the maximum length of the string, just like **fgets ()** does . Blank lines in the file are not ignored, but are returned as a list of one item - an empty line.
Example :

```
$ f = fopen ("file.csv", "r") or die (" Error ");
for ($ i = 0; $ data = fg etscsv ($ f, 1000, ";"); $ i ++) {
  $ num = count ($ data);
  if ($ num == 1 && $ data [0] === "") continue;
  echo "<h3> String number $ i ($ num fields): </h3>";
  for ($ c = 0; $ c <$ num; $ c ++)
    print "[$ c]: $ data [$ c] <br>";
}
fclose ($ f);
```

Position of the current position pointer

feof
End of file pointer.

Syntax:
int feof (int $ f)

Returns true if the end of the file is reached (that is, if the file pointer is positioned past the end of the file).
Example :

```
$ f = fopen ("myfile.txt", "r");
while (! feof ($ f))
{$ str = fgets ($ f);
   // Process the next line $ str
}
fclose ($ f);
```

fseek
Sets the file pointer to a specific position.

Syntax:
int fseek (int $ f, int $ offset, int $ whence = SEEK_SET)

Sets the file pointer to the byte offset *$ offset* (from the beginning of the file, from its end, or from the current position, depending on the *$ whence* parameter). This may not work if the *$ f* descriptor is associated with an HTTP or FTP connection rather than a regular local file.

The *$ whence* parameter specifies where the *$ offset is* calculated from . There are three constants for this in PHP, 0, 1, and 2, respectively:
SEEK_SET
- sets the position starting from the beginning of the file;
SEEK_CUR

- counts down the position relative to the current position;
SEEK_END
- counts down the position relative to the end of the file;
In the case of using the last two constants, the *$ offset* parameter may well be negative (and when using *SEEK_END* it will most likely be negative). This function returns 0 on success and -1 on failure.

ftell
Returns the position of the file pointer.

Syntax :
int ftell (int $ f)

Functions for detecting file types

file_exists
Checks for the existence of the called file.

Syntax:
bool file_exists (string filename)

Returns true if the file named *filename* exists at the time of the call. Use this function with care. For example, the following code is no good from a security standpoint:

```
$ fname = "/ etc / passwd";
if (! file_exists ($ fname))
  $ f = fopen ($ fname, "w");
else
  $ f = fopen ($ fname, "r");
```

The fact is that between the call to **file_exists ()** and the opening of the file in w mode, some time passes, during which another process can intervene and replace the file we are using. This problem comes to the fore when you write a counter script.

The function does not work with remote files, the file must be located in the file system accessible to the server.
Function results are cached, see clearstatcache () function .

filetype
Returns the file type.

Syntax:
string filetype (string filename)

Returns a string that describes the type of file named *filename* . Returns false if no such file exists.
Once called, the string will contain one of the following values:
file is a regular file;
dir - directory;
link - a symbolic link;
fifo - fifo channel;
block - block-oriented device;
char - character-oriented device;
unknown - unknown file type;

is_file
Check **if** a regular file exists.

Syntax :
bool is_file (string filename)

Returns true if *filename* is a regular file .

is_dir
Check for directory existence.

Syntax:
bool is_dir (string filename)

Returns true if *filename* exists.

is_lin k
Check for the existence of a symbolic link to a file.

Syntax:
bool is_link (string filename)

Returns true if *filename* is a symbolic link.

The function does not work under Windows.

is_readable
Checks for the existence of a readable file.

Syntax:
bool is_readable (string filename)

Returns true if the file can be opened for reading.
Typically PHP will access the file with the privileges of the user running the web server (often "nobody").
Safety considerations must be taken into account.

is_writeable
Checks **if** a file exists for writing.

Syntax:
bool is_writeable (string filename)

Returns true if the file can be written to.
Typically PHP will access the file with the privileges of the user running the web server (often "nobody").
Safety considerations must be taken into account.

is_executable
Checks **if the** executable file exists.

Syntax :
bool is_executable (: string filename)

Returns true, if the file *filename* - executable .

is_uploaded_fi le
Check for the existence of a file uploaded using the HTTP POST method.

Syntax:
bool is_uploaded_file (string filename)

Returns true if a file named *filename* was uploaded to the server via HTTP POST.
This is often useful to ensure that users are not maliciously trying to force the script to work with files they should not work with, for example: / etc / passwd.

Defining file parameters

stat
The function gathers together all the information provided by the operating system for the specified file and returns it as an array.

Syntax:
array stat (string $ filename)

This array always contains the following elements with the specified keys:
0 - device;
1 - inode number;
2 - file protection attributes;
3 - the number of synonyms ("hard" links) of the file;
4 - the owner's uid;
5 - group gid;
6 - device type;
7 - file size in bytes;
8 - time of the last access in seconds since January 1, 1970;
9 - time of the last modification of the file content;
10 - time of the last change of file attributes;
11 - block size;
12 - the number of occupied blocks;
This array holds information that is available on Unix systems. On Windows, many fields may be empty.

If *$ filename* specifies a symbolic link rather than a file name, then information about the file to which this link refers (and not about the link) will be returned.

fileatime
Returns the time the file was last accessed.

Syntax:
int fileatime (string filename) The

time is expressed in seconds since January 1, 1970 (Unix timestamp). Returns false if no file is found.
The file's last access time attribute is changed every time the file's data is read. Since this greatly reduces performance when working with files and directories intensively, changing this attribute is often blocked in operating systems, and then the function is useless.
filemtime
Returns the time the file was last modified, or false if the file is missing.

Syntax :
int filemtime (string $ filename)

fi lectime
Returns the creation time of the file.

Syntax :
int filectime (string $ filename)

filesize
Returns the size of the file in bytes, or false if the file does not exist.

Syntax :
int filesize (string $ filename)

touch
Sets the modification time.

Syntax:
int touch (string $ filename [, int $ timestamp])

Sets the modification time of the specified *$ filename* to *$ timestamp* (in seconds since January 1, 1970). If the second parameter is not specified, the current time is assumed. Returns false on error.

If a file with the specified name does not exist, it is created empty.

Functions for working with file names

basename
Separates a filename from a path.

Syntax:
string basename (string $ path)

Separates base name from *$ path*
Examples:
e cho basename ("/ home / somebody / somefile.txt"); // outputs "somefile.txt"
echo basename ("/"); // prints nothing
echo basename ("/."); // prints "."
echo basename ("/./"); // also outputs "."
Function **basename ()** does not check the existence of the file. It just takes the part of the string after the rightmost slash and returns it.
This function handles both forward and backslashes correctly under Windows.

dirname
Highlights the name of a directory.

Syntax:
string dirname (string $ path)

Returns the directory name as extracted from *$ path* . The function is quite "smart" and knows how to highlight non-trivial situations, which are described in the examples:
echo dirname ("/ home / file.txt"); // prints "/ home"
echo dirname (". ./file.txt"); // prints ".."
echo dirname ("/ file.txt"); // prints "/" under Unix,
// "\" under Windows
echo dirname ("/"); // same
echo dirname ("file.txt"); // prints "."
If **dirname () is** simply a filename, it will return ".", Which stands for "current directory".

tempnam
Generates a unique filename in a specific directory.

Syntax:
string tempnam (string $ dir, string $ prefix)

Generates a filename in the *$ dir* directory with *$ prefix* in the name, so that the file created under this name in the future is unique.
To do this , a random number is appended to the *$ prefix* line .
For example, calling tempnam ("/ tmp", "temp") might return / tmp / temp3a6b243c.
If such a name needs to be created in the current directory, pass $ dir = "."

realpath
Converts a relative path to an absolute path.

Syntax:
string realpath (string $ path)

Converts the relative path *$ path* to absolute, i.e. starting from the root.

Example:
echo realpath ("../ t.php"); // e.g. /home/t.php
echo realpath ("."); // displays the name of the current directory
The file specified in the *$ path* parameter must exist, otherwise the function will return false.

Functions for manipulating whole files

copy
Copies the file.

Syntax:
bool copy (string $ src, string $ dst)

Copies a file named *$ src* to a file named *$ dst* . In this case, if the *$ dst* file existed at the time of the call, it is overwritten.
The function returns true if the copy was successful, and false if it failed.

The function does not rename a file if its new name is located in a different file system (on a different mounted system in Unix or on a different disk in Windows).

unlink
Delete file.

Syntax:
bool unlink (string $ filename) Removes the file named *$ filename* . Returns false on failure, true otherwise. It should be noted that the file is deleted only if the number of "hard" links to it has become equal to 0. However, this scheme is specific to Unix systems.

file
Reads a file and splits it line by line.

Syntax:
list file (string $ filename)

Reads the entire file named *$ filename* and returns a list array, each element of which corresponds to a line in the file read.
The disadvantage of this function is that end-of-line characters (usually \ n) are not stripped from the lines of the file, nor are they translated, as is done for text files.

Other functions

ftruncate
Truncates the file.

Syntax:
bool ftruncate (int $ f, int $ newsize)

This function truncates the open file *$ f* to the size of *$ newsize* . Of course, the file must be opened in a writeable mode.
For example, the following code clears the entire file:

ftruncate ($ f, 0);

fflush

Immediately records all changes to the file.

Syntax:
void fflush (int $ f)

Forces PHP to immediately write to disk any changes that were made before with the open file *$ f* . What are these changes? The point is that to improve performance, all file writes are buffered: for example, calling fputs ($ f, "This is a string!") Does not directly write data to disk - first, they go into an internal buffer (usually 8K). As soon as the buffer is full, its contents are sent to disk, and it itself is flushed, and everything is repeated again. A special benefit from buffering is felt in network operations, when it is simply stupid to send data in small portions.

set_file_buffer
Sets the buffer size.

Syntax:
int set_file_buffer (int $ f, int $ size)

This function sets the **above** buffer size for the specified open file *$ f* .
Most often it is used like this:

set_file_buffer ($ f, 0);

The above code turns off buffering for the specified file so that now all data written to the file is immediately sent to disk or to the network.

flock
Blocking a file.

Syntax:
bool flock (int $ f, int $ operation [, int $ wouldblock])

The function sets the blocking mode for the specified open file descriptor *$ f* that the current process would like to get. This mode is specified by the *$ operation* argument and can be one of the following constants:

- LOCK_SH (or 1) - shared lock;
- LOCK_EX (or 2) - exclusive lock;
- LOCK_UN (or 3) - unlock;
- LOCK_NB (or 4) - this constant must be added to one of the previous ones,
  if you don't want the program to hang on
  flock () pending its turn, but immediately returned
  control.

If no wait mode was requested and the lock was not successfully set, the value true will be written to the *optional $ wouldblock* variable parameter .
If an error occurs, the function returns false, and if successful, it returns true.

Functions for working with directories

Manipulating directories

mkdir
Create directory.

Syntax:
bool mkdir (string $ name, int $ perms)

Creates a directory named *$ name* with *perms permissions* . Permissions for directories are specified in the same way as for files. Most often, the value of *$ perms is* set to 0770 (a leading zero is required - it tells PHP that this is an octal constant, not a decimal number).

Example:
```
the mkdir ( " up my _ directory ", 0755);
    // creates a subdirectory in the current directory
mkdir ("/ data");
    // creates a data subdirectory in the root directory
```
If successful, the function returns true, otherwise - false.

rmdir
Delete directory.

Syntax:
bool rmdir (string $ name)

Removes the directory named *$ name* .
The directory must be empty, and its attributes must allow it.
If successful, the function returns true, otherwise - false.

c hdir
Change the current directory.

Syntax:
int chdir (string $ directory);

Changes the current PHP directory to directory. Returns FALSE if it cannot change, TRUE if a change has occurred. The *$ directory* parameter can also specify a relative path from the current directory.

Example:
```
chdir ("/ tmp / data"); // follow the absolute path
chdir ("./ js"); // go to the subdirectory of the current directory
chdir (".."); // go to parent directory
chdir ("~ / data"); // go to / home / user / data (for Unix)
```

getcwd
Full path.

Syntax:
string getcwd ()

This function returns the current directory with respect to which file operations are performed, i.e. returns the full path to the current directory, starting at root (/). If such a path cannot be traced, the call fails and false is returned.

diskfreespace
Defines the free space in the directory

Syntax:
float diskfreespace (string directory);

This function returns the free space in bytes in the directory *directory* , that is, in the corresponding file system or on a disk partition.

Example:
```
$ diskspace = diskfreespace ("/");
// Thus, we have determined the free space in the root directory "/"
```

Working with records

dir
Directory class (pseudo-object-oriented mechanism).

Syntax:
new dir (string directory);

A pseudo-object-oriented mechanism for listing directory files. Opens a directory from directory.
After that, two properties of the object become available: the handle to the directory and the path string indicating which directory is currently being used. These properties are available only if the directory has been opened. The handle property can be used in conjunction with other directory functions such as readdir () , rewinddir (), and closedir () .
Three methods are available for the class: read, return to start, and close (read, rewind and close, respectively).

Example :
```
$ d = dir ("/ etc");
echo "Handle:". $ d-> handle. "<br> \ n";
echo "Path:". $ d-> path. "<br> \ n";
while ($ entry = $ d-> read ()) {// Consecutively output
  echo $ entry. "<br> \ n"; // name of each file,
} // available in the directory
$ d-> close ();
```

closedir
Close the handle of the directory.

Syntax:
void closedir (int dir_handle);

Closes the directory stream denoted by dir_handle. The stream must first be opened with opendir ().

opendir
Open directory descriptor.

Syntax:
int opendir (string path);

Returns a handle to the open directory *path* , which is subsequently used in the closedir () , readdir () , and rewinddir () functions .

readdir
Get the name of the next file in the directory listing.

Syntax:
string readdir (int dir_handle);

Returns the name of the next file in the directory. The file names are returned in unordered order.

Example:
```
<? php
  $ handle = opendir (".");
  echo "Directory handle: $ handle \ n";
  echo "Files: \ n";
  while ($ file = readdir ($ handle)) {
    echo "$ file \ n";
  }
  closedir ($ handle);
?>
```

It should be noted that the function also returns "." and "..". If these values are not required, then they can be excluded as follows:

```
<? php
$ handle = opendir (".");
while ($ file = readdir ($ handle)) {
  if ($ file! = "." && $ file! = "..") {
    echo " Name the file : $ file <br>";
  };
};
closedir ($ handle);
?>
```

re winddir
Reinitialize the directory descriptor.

Syntax:
void rewinddir (int dir_handle);

After calling this function, the readdir () function with the *dir_handle* argument will return the filenames from the beginning in the directory listing.

FTP

Working with an FTP server

ftp_connec t
Connects to an FTP server.

Syntax:
int ftp_connect (string host [, int port])

In **ftp_connect (),** the *host* argument specifies the name of the server to connect to, and the optional *port* argument specifies which port to use (21 by default).
The function returns a handle to the FTP stream, or false on error.

ftp_pasv
Switches passive mode.

Syntax:
int ftp_pasv (int ftp_stream, int pasv)

The **ftp_pasv (**) function switches the connection mode to passive if the *pasv* argument is true. If false, then the connection mode will be active.
In the passive mode, data transfer is initiated by the client, and in the active mode, by the server (this is sometimes necessary when blocking ports on the client).
The function will return true or false on error.

ftp_login Logs
in to the FTP server.

Syntax:
int ftp_login (int ftp_stream, string username, string password)

The **ftp_login ()** function **logs** in to the system as *username* with the password *passw ord* . Returns true or false on error.

ftp_quit Quit an
FTP session.

Syntax :

int ftp_quit (int ftp_stream)

ftp_pwd
Determines the current directory.

Syntax:
int ftp_pwd (int ftp_stream)

This function returns the current directory of the FTP server, or false on error.

ftp_cdup Changes
to the root directory.

Syntax:
int ftp_cdup (int ftp_stream)

The function returns true or false on error.

ftp_chdir Changes
to the directory.

Syntax:
int ftp_chdir (int ftp_stream, string directory)

The function returns true or false on error.

ftp_mkdir Creates
a directory.

Syntax:
int ftp_mkdir (int ftp_stream, string directory)

The function returns the name of the created directory or false on error.

ftp_rmdir
Deletes a directory.

Syntax :
int ftp_rmdir (int ftp_stream, string directory)

Function true or false on error .

ftp_nlist Get
a directory listing.

Syntax:
int ftp_nlist (int ftp_stream, string directory)

The **ftp_nlist ()** function returns an array of filenames or false on error.

ftp_rawlist
Get a detailed listing of a directory.

Syntax:
int ftp_rawlist (int ftp_stream, string directory)

The **ftp_rawlist ()** function executes the FTP LIST command, and returns its results in an array, where each item corresponds to a string of text as is. The system type identifier returned by **ftp_systype ()** can be used to determine how the results should be interpreted.

ftp_systype
Returns the system identifier of the FTP server type.

Syntax:
int ftp_systype (int ftp_stream)

The function returns a string value or false in case of an error.

Working with files

ftp_get Downloads
from an FTP server.

Syntax:
int ftp_get (int ftp_stream, string local_file, string remote_file, int mode)

The **ftp_get ()** function downloads a file called *remote_file* from the FTP server and stores it locally as *local_file* . The *mode* parameter sets the file transfer mode and can be FTP_ASCII (text) or FTP_BINARY (binary, binary).
The function returns true or false on error.

ftp_fget Downloads
and writes a file.

Syntax:
int ftp_fget (int ftp_stream, string fp, string remote_file, int mode)

The **ftp_fget (**) function downloads a file called *remo te_file* from the FTP server and stores it in a file that has a descriptor *fp* . The *mode* parameter sets the file transfer mode and can be FTP_ASCII (text) or FTP_BINARY (binary, binary).
The function returns true or false on error.

ftp_put Uploads
a file to an FTP server.

Syntax :
int ftp_put (int ftp_stream, remote_file: string, local_file: string, int mode)

function **ftp_put ()** loads a file on FTP- server under the name *remote_file* . The *mode* parameter sets the file transfer mode and can be FTP_ASCII (text) or FTP_BINARY (binary, binary). The function returns true or false on error.

```
$ upload = ftp _ put ($ ftp _ id , " C : \\ file . txt ", "/ file . txt ", FTP _ ASCII );
```

ftp_fput
Reads and uploads a file to the FTP server.

Syntax:
int ftp_fput (int ftp_stream, string remote_file, string fp, int mode)

The **ftp_fput (**) function reads the open file with descriptor *fp* to the end and uploads this file to the FTP server as *remote_file* . The *mode* parameter sets the file transfer mode and can be FTP_ASCII (text) or FTP_BINARY (binary, binary).
The function returns true or false on error.

ftp_size
Determines the size of the file.

Syntax:
int ftp_size (int ftp_stream, string remote_file)

The **ftp_siz e (**) function returns the size of the file specified in the *remote_file* parameter in bytes, or -1 on error.
Not all servers support this feature.

ftp_mdtm
Returns the time the file was last modified.

Syntax:
int ftp_mdtm (int ftp_stream, string remote_file)

The **ftp_mdtm (**) function returns the time, last modified, in Unix format, or -1 on error.
This function does not work with directories.

ftp_rename Renames a
file.

Syntax :
int ftp_rename (int ftp_stream, string from, s tring to)

The **ftp_rename ()** function renames the file *from* to *to* . The function returns true or false on error.

ftp_delete
Deletes a file from the server.

Syntax:
int ftp_delete (int ftp_stream, string path)

The **ftp_delete ()** function deletes the file specified in the *path* parameter .
The function returns true or false on error.

ftp_site Executes a
SITE command on the server.

Syntax:
int ftp_site (int ftp_stream, string cmd)

The **ftp_site ()** function sends the cmd command to the server.
Because SITE commands are not standardized and may vary. They are usually useful for changing file permissions and group ownership.
The function returns true or false on error.

IMAP functions

In order for these functions to work, you must compile PHP with the --with-imap flag.
This flag requires the c-client library to be installed. The latest version can be obtained from ftp://ftp.cac.washington.edu/imap/ .
Then copy c-client / c-client.a to / usr / local / lib or some other directory specified in the path, then copy c-client / rfc822.h, mail.h and linkage.h to / usr / local / include or another directory with include files.

Despite the name of the module, the functions it contains allow you to perform many other useful operations as well, beyond the simple use of IMAP. The underlying C client library also supports NNTP, POP3, and local mailbox access methods.

imap_append

Adds a text message to the specified mailbox.

Syntax:
int imap_append (int imap_stream, string mbox, string message, stringflags);

Returns true on success or false otherwise.
imap_append () adds a text message to the specified mbox mailbox. If optional flags are specified, also writes to mailbox and flags. When communicating with the Cyrus IMAP server, use "\ r \ n" instead of "\ n" as line delimiters, otherwise the action will not be performed.

imap_base64
Decodes **BASE64** encoded text.

Syntax:
string imap_base64 (string text);

Imap_base64 () function decodes text in BASE-64 format. The decoded message is returned as a string.

imap_body
Reads the body of the message.

Syntax:
string imap_body (int imap_stream, int msg_number, int flags);

The imap_body () function returns the body of the message with the p / p number msg_number in the current mailbox.
Optional flags are bit masks from
FT_UID - Message number msgno is the UID of
FT_PEEK message - Don't set the \ Seen flag if it is not already set.
FT_INTERNAL - The returned string is in internal format and cannot be canonicalized with CRLF.

imap_check
Checks the current mailbox.

Syntax:
array imap_check (int imap_stream);

Returns information about the current mailbox. Returns FALSE on failure.
The imap_check () function checks the status of the current mailbox on the server and returns information in an object with the following properties:
Date: date of the message
Driver:
Mailbox driver : name of the mailbox
Nmsgs: number of messages
Recent: number of recently arrived messages

imap_close
Closes the IMAP stream.

Syntax :
int imap_close (int imap_stream, int flags);

Closes the imap stream . The optional CL_EXPUNGE flag causes the messages marked for deletion to be erased on close.

imap_createmailbox
Creates a new mailbox.

Syntax:
int imap_createmailbox (int imap_stream, string mbox);

imap_createmailbox () creates a new mailbox specified in mbox. Returns true on success and false on failure.

imap_delete
Marks a message in the current mailbox for deletion.

Syntax :
int imap_delete (int imap_stream, int msg_number);

Returns true.
Returns true The imap_delete () function marks the message specified by msg_number for deletion. The actual deletion of messages is done with the imap_expunge () function.

imap_deletemailbox
Deletes a mailbox.

Syntax:
int imap_deletemailbox (int imap_stream, string mbox);

Returns true on success and false otherwise.

imap_expunge
Deletes all messages marked for deletion.

Syntax:
int imap_expunge (int imap_stream);

imap_expunge () deletes all messages marked for deletion with imap_delete ().
Returns true.

imap_fetchbody Fetch a
simple section of the message body.

Syntax:
string imap_fetchbody (int imap_stream, int msg_number, int part_number, flags flags);

This function causes the verbose section of the specified message to be retrieved as a text string. The section is a dotted-separated string of integers that indicate the body parts of the message in the IMAP4 parts list. Body parts are not decoded by this function.
An optional parameter to imap_fetchbody () is a bitmask from
FT_UID - msgono is the FT_PEEK
UID - do not set the \ Seen flag if it is not set
FT_UID - the returned string is written in an internal format that cannot be canonicalized with CRLF

imap_fetchstructure
Reads the structure of a simple message.

Syntax:
array imap_fetchstructure (int imap_stream, int msg_number);

array imap_fetchstructure (int imap_stream, int msg_number); This function forces to retrieve all information about the structure of the message with the number msg_number. The return value is an object with the following elements:
type, encoding, ifsubtype, subtype, ifdescription, description, ifid, id, lines, bytes, ifparameters type, encoding, interface subtype, subtype, interface description, description, interface identifier, lines, bytes, parameters interface
The function also returns an array of objects called parameters []. This object has the following properties:
attribute, value
attribute, value

In the case of a message from several parts, the function also returns an array of objects of all properties called parts [].

imap_head er
Reads the message header.

Syntax:
object imap_header (int imap_stream, int msg_number, int fromlength, int subjectlength, int defaulthost);

This function returns an object of various header elements
remail, date, Date, subject, Subject, in_reply_to, message_id, newsgroups, followup_to, references
toaddress (full string To: string up to 1024 characters)
to [] (returns an array of objects from string To, contains :)
personal
adl
mailbox
host
fromaddress (full line From: line up to 1024 characters)
from [] (returns an array of objects from line From, contains :)
personal
adl
mailbox
host
ccaddress (full line Cc: line up to 1024 characters)
cc [ ] (returns an array of objects from string Cc, contains)
personal
adl
mailbox
host
bccaddress (full Bcc string: string up to 1024 characters)
bcc [] (returns an array of objects from string Bcc, contains :)
personal
adl
mailbox
host
reply_toaddress (full string Reply_to : string up to 1024 characters)
reply_to [] (returns an array of objects from the Reply_to string, contains :)
personal
adl
mailbox
host
senderaddress (full string Sender: string up to 1024 characters)
sender [] (returns array of objects from the Sender line, contains :)
personal
adl
mailbox
host
return_path (full Return-path line: up to 1024 characters)
return_path [] (returns an array of objects from the Return_path line, contains :)
personal
adl
mailbox
host
udate (date of the message in unix time format)
fetchfrom (From string formatted to fromlength characters)
fetchsubject (Subject string formatted to subjectlength characters)

imap_headers

Returns the headers of all messages in a mailbox.

Syntax:
array imap_headers (int imap_stream);

Returns a string array of headers information. One array element per message.

imap_listmailbox
Reads a list of mailboxes.

Syntax:
array imap_listmailbox (int imap_stream, string ref, string pat);

Returns an array containing the mailbox names.

imap_listsubscribed
Lists all subscribed mailboxes.

Syntax:
array imap_listsubscribed (int imap_stream, string ref, string pattern);

Returns an array of all mailboxes to which you are subscribed. The ref and pattern arguments specify the starting location to start the search from and the pattern that mailbox names must match.

imap_mail_copy
Copies the specified messages to the mailbox.

Syntax:
int imap_mail_copy (int imap_stream, string msglist, string mbox, int flags);

Returns true on success and false otherwise.
Copies the mails specified with msglist to the mbox mailbox. msglist is a range, not just a message number.
Flags are bit masks from
CP_UID - numbers in the sequence contain
UIDs CP_MOVE - after copying, delete messages from the current mailbox

imap_mail_move Moves the
specified messages to the mailbox.

Syntax:
int imap_mail_move (int imap_stream, string msglist, string mbox);

Moves mails specified with msglist to the mbox mailbox. msglist is a range, not just a message number.
Returns true on success and false otherwise.

imap_num_msg
Returns the number of messages in the current mailbox.

Syntax:
int imap_num_msg (void);

Returns the number of messages in the current mailbox.

imap_num_rec ent
Returns the number of recently received messages in the current mailbox.

Syntax :
int imap _ num _ recent (int imap _ stream);

imap_open

Connecting to the server (opening the mailbox).

Syntax:
int imap_open (string mailbox, string username, string password [, int flags]);

Function **imap_open ()** returns a handle IMAP mailbox (the connection handle to the IMAP server), or false on error.
This function can be used to open streams to POP3 and NNTP servers, but in this case some functions will not be available.

The *mailbox* argument specifies the server name and the path to the mailbox. The server name should be enclosed in curly braces "{" and "}", inside which should contain: the server name (or its IP address), possibly indicating the protocol (which begins with a slash "/") and the port number In order to connect to the IMAP server on port 143 on the local machine do the following:
$ mbox = imap_open ("{localhost: 143} INBOX", "user_id", "password");
To connect to the POP3 server on port 110 on the local server use:
$ mbox = imap_open ("{localhost / pop3: 110} INBOX", "user_id", "password");
To connect to the NNTP server on port 119 on the local server use:
$ nntp = imap_open ("{localhost / nntp: 119} comp.test", "", "");
To connect to a remote server, replace "localhost" with the name or IP address of the server you want to connect to.
Options - bitmask from
OP_READONLY - Open mailbox in read-only mode
OP_ANONYMOUS - Do not use or update .newsrc when using news
OP_HALFOPEN - For IMAP and NNTP, connects, but does not open mailbox
CL_EXPUNGE - Automatically clear mailbox on close

imap_ping
Checks the IMAP stream for health.

Syntax:
int imap_ping (int imap_stream);

Returns true if the stream is still healthy and false otherwise.
The imap_ping () function checks the stream for health. He can also check new mail; this is the preferred method for periodically checking for new mail and for keeping remote servers alive.

imap_renamemailbox
Renames an old mailbox to a new one.

Syntax:
int imap_renamemailbox (int imap_stream, string old_mbox, string new_mbox);

This function renames an old mailbox to a new one.
Returns true on success and false otherwise.

imap_reopen Reopens the
IMAP stream to a new mailbox.

Syntax:
int imap_reopen (string imap_stream, string mailbox, string [flags]);

Returns true on success and false otherwise.
This function reopens the specified stream to a new mailbox.
Options - bitmask from
OP_READONLY - Open mailbox in read-only mode
OP_ANONYMOUS - Do not use or update .newsrc when working with news
OP_HALFOPEN - For IMAP and NNTP, connects but does not open mailbox
CL_EXPUNGE - Clears mailbox on close

imap_subscribe
Subscribes to a mailbox.

Syntax:
int imap_subscribe (int imap_stream, string mbox);

Returns true on success and false otherwise.

imap_undelete Unchecks a
message marked for deletion.

Syntax:
int imap_undelete (int imap_stream, int msg_number);

This function unchecks a message marked for deletion with imap_delete ().
Returns true on success and false otherwise.

imap_unsubscribe
Unsubscribes from a mailbox.

Syntax:
int imap_unsubscribe (int imap_stream, string mbox);

Returns true on success and false otherwise.

imap_qprint
Converts a quoted-printable string to an 8-bit string.

Syntax:
string imap _ qprint (string string); Returns an 8-bit (binary) string.

imap_8bit
Converts an 8-bit string to quoted-printable format.

Syntax :
string imap_8bit (string string);

Returns a string in the format quoted-printable.

imap_binary
Converts an 8-bit string to base64 format.

Syntax:
string imap_binary (string string);

Returns a string in base64 format.

imap_scanmailbox
Reads the list of mailboxes, searches for mailbox names.

Syntax:
array imap_scanmailbox (int imap_stream, string string);

Returns an array containing mailbox names that have string in their name.

imap_mailboxmsginfo
Gets information about the current mailbox.

Syntax:
array imap_mailboxmsginfo (int imap_stream);

Returns information about the current mailbox. FALSE on failure.
The imap_mailboxmsginfo () function checks the status of the current mailbox on the server and returns information in an object with the following properties:
Date: message date
Driver:
Mailbox driver : mailbox name
Nmsgs: number of messages
Recent: number of recently received messages
Unread: number of unread messages
Size: size mailbox

imap_rfc822_write_address
Returns a properly formatted email address.

Syntax:
string imap_rfc822_write_address (string mailbox, string host, string personal);

Returns a properly formatted email address for the given mailbox, host and personal information.
imap_rfc822_parse_adrlist Parses the
address bar.

Syntax:
string imap_rfc822_parse_adrlist (string address, string default_host);

This function parses the address bar and returns an array of objects for each address.
There are 4 types of objects:
mailbox - mailbox name (username)
host - host name
personal - personal name
adl - path to the source domain

imap_setflag_full
Sets flags on messages.

Syntax:
string imap_setflag_full (int stream, string sequence, string flag, string options);

This function forces to add the specified flag to the set of message flags in the specified sequence.
options is a bit mask from ST_UID Sequence
arguments contain UIDs instead of numbers

imap_clearflag_full
Clears message flags.

Syntax:
string imap_clearflag_full (int stream, string sequence, string flag, string options);

This function causes the flags to be removed from the message flag set in the specified sequence.
options is a bit mask from ST_UID Sequence
arguments contain UIDs instead of numbers

imap_sort Sorts
messages in the current mailbox.

Syntax:
string imap_sort (int stream, int criteria, int reverse, int options);

Returns an array of message numbers sorted by this parameter
Rev must be equal to 1 if sorting in reverse order
Sorting criteria (only one must be specified): SORTDATE - by message date
SORTARRIVAL - by date of receipt
SORTFROM - by the From
SORTSUBJECT field - by message subject
SORTTO - by field To
SORTCC - by field cc
SORTSIZE - by
option size - bit mask from
SE_UID - Return UIDs instead of sequence numbers
SE_NOPREFETCH - Do not fetch previously found messages

imap_fetchheader
Returns the header of the message.

Syntax:
string imap_fetchheader (int imap_stream, int msgno, int flags);

This function causes the full, unfiltered RFC 822 header of the specified message to be retrieved as a text string.
Options:
FT_UID msgno is the UID
FT_INTERNAL The returned string is written in internal format without any attempt to canonize it using CRLF FT _ PREFETCHTEXT RFC 822. The text must be parsed beforehand. This will help to avoid urgent delays if you need to extract the full text of the message (for example, in the operation "save to local file")

imap_uid
This function returns the UID of the given message number in sequence.

Syntax :
string imap _ uid (string mailbox , int msgno);

SNMP functions

snmpget
Gets an SNMP object.

Syntax:
int snmpget (string hostname, string community, string object_id);

Returns the value of the SNMP object on success and false on error.
The snmpget () function is used to read the SNMP value of the object specified by object_id.
The SNMP agent is specified by the hostname hostname and the read group is specified by the community parameter.
snmpget ("127.0.0.1", "public", "system.SysContact.0")

snmpwalk
Receives all SNMP objects from the agent.

Syntax:
array snmpwalk (string hostname, string community, string object_id);

Returns an array of SNMP object values starting with object_id and false on error.
The snmpwalk () function is used to read all values from the SNMP agent specified by the hostname parameter.
Community defines a reading group for the agent.

A null object_id is taken as the root of the SNMP object tree, and all objects under that tree are returned as an array.
If object_id is specified, then all SNMP objects below this object are returned.
$ a = snmpwalk ("127.0.0.1", "public", "");
The above function call will return all SNMP objects from the SNMP agent fired on the local host.
All values can be traversed using a loop:

```
for ($ i = 0; $ i < count ($ a ); $ i ++) {
        echo $ a [$ i];
}
```

Vmailmgr functions

These functions require the QMAIL packages (www.qmail.org) and vmailmgr Bruce Guenter
http://www.qcc.sk.ca/~bguenter/distrib/vmailmgr/
For all functions, the following two variables are defined as: string vdomain - your domain name virtual domain (vdomain.com), the basepwd string is the password for the "real" user that supports virtual users. Only up to 8 characters are recognized in the password for virtual users.
The status is returned for all response function values in response.h
O ok
1 bad
2 error
3 connection error

```
<? php
dl ("php3_ vmailmgr.so"); // load the shared library
$ vdomain = "vdomain.com";
$ basepwd = "password";
?>
```

vm_adduser
Adds a new virtual user with a password.

Syntax:
int vm_adduser (string vdomain, string basepwd, string newusername, string newuserpassword);

Adds a new virtual user with a password. newusername is the mail login name and newuserpassword is the password for this user.

vm_addalias
Adds a new alias for the virtual user.

Syntax:
int vm_addalias (string vdomain, string basepwd, string username, string alias);

Adds an alias to the virtual user. username is the name of the mail login and alias is the alias for this user.

vm_passwd
Changes the password for virtual users.

Syntax:
int vm_passwd (string vdomain, string username, string password, string newpassword);

Changes the password of virtual users. username is the mail login name, password is the user's old password, and newpassword is the new password.

vm_delalias
Deletes an alias . **Syntax :** int vm_delalias (string vdomain, string basepwd, string alias);

vm_deluser
Deletes the virtual user alias.

Syntax :
int vm_deluser (string vdomain, string username);

Network functions

ip2long Convert an
IPv4 address string to a number.

Syntax:
int ip2long (string ip_address);

The **ip2long ()** function returns a four-byte numeric representation of an IP v4 address from a string (dot-separated numbers, for example: "127.0.0.1").

```
// get the IP address of the host
$ ip = gethostbyname ("www.php.net");
echo "The following URLs are equivalent: <br>";
echo "http://www.php.net/, http: //". $ ip.
    "/, and http: //" .ip2long ($ ip). "/ <br>";
```

long2ip Converts a
number to an IP v4 address string.

Syntax:
string long2ip (int proper_address);

The **long2ip ()** function returns the string representation of the IP address (in the format: "aaa.bbb.ccc.ddd") from the numeric representation.

gethostbyaddr
Returns the hostname that matches the given IP address.

Syntax:
string gethostbyaddr (string ip_address); **Gethostbyaddr ()**
function returns the domain name of the host specified by its IP address. The argument specifies the IP address in string format. Returns *ip_address on error* . It should be noted that the function does not guarantee that the resulting name will actually match reality. It only polls the host at *ip_address* and asks him to provide his name. The owner of the host can thus transfer whatever he pleases.

```
echo gethostbyaddr ("127.0.0.1");
```

gethostby name
Returns the host's IP address.

Syntax:
string gethostbyname (string hostname);

The **gethostbyname ()** function takes the host's domain name as parameters and returns its IP address. If the address could not be determined, the function returns *hostname* .

gethostbynamel
Returns a list of host IP addresses.

Syntax:
array gethostbynamel (string hostname);

Several IP addresses can correspond to one domain name at once, and in case of heavy server load, the DNS server itself chooses to which IP address to redirect the request. He chooses the most rarely used address. The **gethostbynamel ()** function returns not one, but all the IP addresses of the host named *hostname* .

It is worth noting that there are many virtual hosts on the Internet that have different domain names, but the same IP address. Thus, if the following command sequence for an existing host with an IP address *ip* always prints the same address:

```
$ host = gethostbyaddr ($ ip);
echo gethostbyname ($ host);
```

then a similar sequence for a domain with a DNS name *$ host* , on the contrary, may print not the same name, but a different one:

```
$ ip = gethostbyname ($ host);
echo gethostbyaddr ($ ip);
```

getprotobyname
Determines the port number used by the protocol.

Syntax :
int getprotobyname (string name);

getprotobynumber
Determines the port protocol.

Syntax :
string getprotobynumber (int number);

getservbyname
Determines the protocol of the Internet service.

Syntax:
int getservbyname (string service, string protocol);

This function returns the port number that the service uses the **service,** .
The **protocol** argument specifies the type of protocol, TCP or UDP.

```
echo getservbyname ("HTTP", "TCP"); // can print 80
```

getservbyport
Determines the Internet service that uses the specified port.

Syntax:
string getservbyport (int port, string protocol);

Here, in the **protocol** argument, you must specify the type of protocol - TCP or UDP.

```
echo getservbyport (21, " TCP "); // will output: ftp
echo getservbyport (23, " TCP "); // will output: telnet
```

c heckdnsrr Checks the
DNS record.

Syntax:
int checkdnsrr (string host [, string type]);

This function queries the DNS server to find the records that are available for **host** . If records of type *type* were found , the function returns true. Otherwise and on error, false.

The *type* argument can be:

- A
- MX (default)
- NS
- SOA
- PTR
- CNAME
- ANY

The *host* argument can be an IP dotted string or a hostname.

getmxrr Get the
MX record for the internet host.

Syntax:
int getmxrr (string hostname, array mxhosts [, array weight]);

The **getmxrr ()** function initiates a DNS lookup for the MX (domain mail server) record for host *hostname* .
If the record is found, it returns true, if not, then false.

The MX records are listed in the *mxhosts* array . If a *weight* array is specified , it is filled with additional information about the entries.

Part 4. Control functions

Tracking and handling errors

Introduction

PHP has the following types of errors and warnings:

Value	Constant	Description
1	E_ERROR	Fatal runtime error.
2	E_WARNING	Runtime warning.
4	E_PARSE	Runtime interpretation message.
8	E_NOTICE	Simple runtime message.
sixteen	E_CORE_ERROR	Fatal error during PHP initialization.
32	E_CORE_WARNING	Initialization warning.
64	E_COMPILE_ERROR	Fatal compilation error.
128	E_COMPILE_WARNING	Compilation warning.
256	E_USER_ERROR	User-defined errors.
512	E_USER_WARNING	User-defined warnings.
1024	E_USER_NOTICE	User-defined messages.
2047	E_ALL	All listed messages.

The specified values in the form of numbers or constants can be combined to form a bit mask of errors that must be reported during script execution. Bitwise operators are used for combining, but only "|", "~", "!" Are recognized in the php.ini configuration file. and "&".

In PHP 4, messages of the form E_ALL & ~ E_NOTICE are allowed by default, that is, everything should be reported except for normal messages. You can override this setting with the **error_reporting ()** configuration file parameter (it can also be specified in the Apache server configuration files).

If, when calling a function, you specify the "@" symbol in front of its name, then if an error occurs in this function, a message about it will not be displayed.

Currently, the ignore error operator even blocks the issuance of critical error messages, in which case the script terminates ahead of schedule.

If the track_errors configuration parameter is enabled, the error message is stored in the $ php_errormsg global variable.

```
<?
// user-defined error handler
function userErrorHandler ($ errno , $ errmsg , $ filename , $ linenum , $ vars ) {
  // time when the error occurred
$ dt = date ("Ymd H: i: s (T)");
$ errortype = array (
  1 => "Error",
  2 => "Warning",
  4 => "Parsing Error",
  8 => "Notice",
  16 => "Core Error",
  32 => "Core Warning",
  64 => "C ompile Error",
  128 => "Compile Warning",
  256 => "User Error",
  512 => "User Warning",
  1024 => "User Notice"
  );

$ err. = " time ($ dt), error number ($ errno),";
$ err. = " error type (". $ errortype [$ errno]. "):";
$ err. = "\" $ errmsg \ ". file \" $ filename \ ", line (";
```

```
  $ err. = $ linenum. ") \ n";

  $ user_errors = array (E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE);
  if (in_array ($ errno, $ user_errors))
  // issue a message for user errors
  echo $ err;

  // save the error event to the system log
  error_log ($ err, 3, "/usr/local/php4/error.log");
}

// set error control level and handler
error_reporting (0); // do not display PHP messages
$ old_error_handler = set_error_handler ("userErrorHAndler");

// undefined constant raises a warning
$ t = _ NOT _ DEFINED _ CONSTANT ;

trigger_error (" My mistake ", E_USER_ERROR);
trigger_error (" My warning ", E_USER_WARNING);

?>
```

Error handling functions

error_log
Send an error message.

Syntax:
int error_log (string message, int message_type [, string destination [, string extra_headers]]) The

message sent by this function can be sent to the web server syslog, TCP protocol, or file.
The first argument, *message,* specifies the content of the message itself. The second argument *message_type* is where it should be sent.
The purpose is indicated by the following values:

- 0 - The message is written to the system event log (file) according to the setting of the **error_log** configuration parameter .
- 1 - The message is sent by email to the address specified in the *destination* argument . This is the only message type that uses the fourth parameter *extra_headers* , which allows you to specify additional headers (as in the **mail ()** function).
- 2 - The message is sent through the debug connection. This is only possible if the remote debugging option has been enabled in the config file. For this, the host address (name or its IP address) and the socket port that will receive debug messages must also be defined. This can be specified in the *destination* argument or configuration parameters.
- 3 - *message is* appended to the end of the *destination* file .

```
if (! Ora_London ($ username, $ password)) {
  error_log ("Oracle server is unavailable!", 0);
};

if (! ($ foo = allocate_new_foo ()) {
  error_log ("FOO cannot be selected!", 1, "operator@mydomain.ru");
}

// other ways of calling error_log ():
error_log ("We have an error!", 2, "127.0.0.1:7000");
error_log (" We have an error !", 2, "localhost");
error_log (" We have an error !", 3, "/var/tmp/my-errors.log");
```

error_reporting
Sets the types of reported errors.

Syntax:
int error_reporting ([int level])

The **error_reporting ()** function returns the previous setting for the type of errors reported. If an argument is specified, then overrides it again. The argument can be a constant, number, or bit mask. Try to use constants instead of numeric values to maintain compatibility with future PHP versions.

```
error_reporting (2039); // in the PHP equivalent of E_ALL ^ E_NOTICE
error_reporting (E_ALL ^  E_NOTICE); // setting of default
error_reporting (0); // disable error messages
// general runtime errors
error_reporting (E_ERROR | E_WARNING | E_PARSE);
// also inform about the unknown variables
error_reporting (E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
error_reporting (E_ALL); // report all errors
```

restore_error_handler
Restore the previous error handler.

Syntax:
void restore_error_handler ()

This function sets the error handler function to the one that was before the last call to **set_error_handler ()** . The previous handler can be a previously installed custom handler or a built-in PHP handler.

trigger_error
Generate an error.

Syntax:
void trigger_error (string error_msg [, int error_type])

Explicitly calls the function set to handle errors, and is typically used in conjunction with an error handler. The function is only capable of generating user-defined error types (E_USER family of constants), and by default, if *error_type is* not specified , it is considered E_USER_NOTICE.

It is possible to construct complex constructions for generating and handling errors and exceptions.

```
if (assert ($ divisor == 0))
 trigger_error ( " You can not share to 0", E_USER_ERROR);
```

user_error
Synonym for **trigger_error ()** function . **Syntax :** void user_error (string error_msg [, int error_type])

Installing a custom error handler

set_error_handler
Set a custom error handler.

Syntax:
string set_error_handler (string error_handler)

The function returns the name of the function previously defined as an error handler (or FALSE on error) and sets the function with the name specified in the *error_handler* argument as a new handler .

Typically a custom error handler is paired with a **trigger_error ()** function that generates an error. This can be used (similar to the analogous construction of dealing with exceptions in C) to release allocated resources (for example, delete generated files) if the script cannot complete normally.

A function set as an error handler must take five parameters (the last three are optional and may not be processed):

- error code
- a string describing the error
- the name of the script in which the error occurred
- line number of the script containing the error
- context (an array containing the values of variables at the time of the error)

```
<?
// define custom error constants
define ( FATAL , E _ USER _ ERROR );
define (ERROR, E_USER_WARNING);
define (WARNING, E_USER_NOTICE);

// set what errors should be handled in the script
error_reporting (FATAL | ERROR | WARNING);

// custom error handler
function myErrorHandler ($ errno, $ errstr, $ errfile, $ errline) {
  switch ($ errno) {
  case FATAL:
    echo "<b> Fatal error </b> [$ errno] $ errstr <br> \ n";
    echo " at line : $ errline of file :". $ errfile;
    echo ", PHP" .PHP_VERSION. "(" .PHP_OS. ") <br> \ n";
    echo "Aborting ... <br> \ n";
    exit -1;
    break;
  case ERROR:
    echo "<b> Error </b> [$ errno] $ errstr <br> \ n";
    break;
  case WARNING:
    echo "<b> Warning </b> [$ errno] $ errstr <br> \ n";
    break;
  default:
    echo " Unknown error type : [$ errno] $ errstr <br> \ n";
  }
}

// function to check error handling
// (scaling the array
function scale_by_log ($ vect, $ scale) {
  if (! is_numeric ($ scale) || $ scale <= 0)
    trigger_error ("log (x) cannot be calculated for x <= 0. ",
                "(x = $ scale)", FATAL);
  if (! is_array ($ vect)) {
    trigger_error (" Array required ", ERROR);
    return null;
  }
  for ($ i = 0; $ i <count ($ vect); $ i ++) {
    if (! is_numeric ($ vect [$ i]))
      trigger_error ("Item ($ i) is not a number and
                  its value is considered 0 ", WARNING);
    $ temp [$ i] = log ($ scale) * $ vect [$ i];
```

```
  }
  return $ temp;
}

// install a custom error handler
$ old_error_handler = set_error_handler ("myErrorHandler");

$ a = array (2,3, "foo", 5.5,43.3,21.11);
print_r ($ a);

$ b = scale_by_log ($ a, M_PI); // warning is issued here
echo "Array scaled by logarithm (pi):";
print_r ($ b);

$ c = scale_by_log ("not array", 2,3); // error here
var _ dump ($ c );

$ d = scale_by_log ($ a, -2.5); // critical error here

echo "Continuing script ...";
?>
```

When the script is executed, the output will be as follows:

```
Array
(
  [0] => 2
  [1] => 3
  [2] => foo
  [3] => 5.5
  [4] => 43.3
  [5] => 21.11
)
<b> Warning </b> [1024] Element (2) is not a number,
  and its value is considered 0 <br>
Array scaled by logarithm (pi): Array
(
  [0] => 2.2894597716988
  [1] => 3.4341896575482
  [2] => 0
  [3] => 6.2960143721717
  [4] => 49.566804057279
  [5] => 24.165247890281
)
<b> Error </b> [512] Array required <br>
NULL
<b> Critical error </b> [256] it is impossible to calculate log (x) for x <= 0,
  (x = -2.5) <br>
  at line: 37, file E: \ www \ exampl.php, PHP 4.0.5 (WINNT) <br>
Aborting ... <br>
```

Keep in mind that the PHP standard handler is not used when installing a custom error handler. Setting **error_reporting () will** also have no effect, and the custom handler must be able to handle all kinds of errors (the value of **error_reporting ()** can be traced and acted upon). Note that the error code will be 0 if the error occurred in a function for which error output was blocked by the "@" operator.

Also remember that you must explicitly end the script in the handler (for example, using the **die ()** function), if, of course, there is a need for it. If the error handler exits with return, then script execution continues where the error occurred (that is, the instructions that follow the one in which the error occurred) are executed.

Session management

Why sessions are needed. How sessions work

Why sessions are needed .

A session is a mechanism that allows you to store some data that is individual for each user (for example, his name and account number) between script runs.

There is one class of tasks in Web programming that can cause quite a lot of problems if written head-on. We are talking about the weak side of CGI - the inability to run a program for a long time, while allowing it to exchange data with users.

Imagine that we are writing a form, but it has so many fields that it would be silly to put them on one page. We need to break down the process of filling out a form into several stages, or stages, and present them as separate HTML documents.
For example, in the first document with a dialogue, the user may be prompted for his first and last name, in the second - information about his place of residence, and in the third - a credit card number. At any time, you can go back a step to correct certain data. Finally, if everything is in order, the accumulated information is processed - for example, placed in a database.

Implementing such a scheme turns out to be a rather nontrivial problem for Web applications. Indeed, we will have to store all previously entered data in some storage, which should be canceled if the user suddenly changes his mind and leaves the site. You can use serialization functions and files for this. However, we solve only half of the problem with them: we need to somehow bind a specific user to a specific temporary storage. Indeed, suppose we did not. Then, if at the time of filling out a form by one user, another comes to the site and also tries to enter his data, you will get a beleberd.
All these problems are solved with sessions.

Sessions work mechanism .

To begin with, there must be a mechanism for PHP to identify each user who runs the script. That is, the next time you start PHP, you need to unambiguously determine who launched it: the same person, or another. This is done by assigning a so-called unique *session identifier to the* client . To make this identifier available each time the script is run, PHP places it in the browser cookies. Now, knowing the identifier (hereinafter referred to as the SID), PHP can determine in which file on the disk the user data is stored.

A little about how to save a variable (necessarily global) in a session. To do this, we must register it using a special function. After registering, we can be sure that the next time the script is run by the same user, it will receive the same value that it had at the previous program exit. This is because PHP automatically saves all the variables registered in the session to temporary storage when the script ends. Of course, you can invalidate a variable at any time - delete it from the session, or destroy all session data altogether.

Where is the staging storage that PHP uses? Generally speaking, you are free to specify this by writing the appropriate functions and registering them as session handlers. However, this is not necessary: PHP already has default handlers that store data in files. If you are not going to create something special, they are fine for you.

Session initialization and variable registration

session_start
This function initializes the session mechanism for the current user who started the script.

Syntax:
void session _ start ()

- If a visitor launches the program for the first time, cookies with a unique identifier are installed for him,

and a temporary storage is created associated with this identifier.
- Determines which store is associated with the current user ID.
- If there are any variables in the storage, their values are restored. More precisely, global variables are created that were saved in the session when the script was terminated earlier.

It should be noted that if you set the *session.auto_start = 1* mode in the PHP settings , then the initialization function is called automatically when the script starts. You also need to make sure that there is no output to the browser before our function - otherwise PHP will not be able to set the SID for the user.
The function always returns true.
session_register
Tells PHP to store this or that variable in the session.

Syntax:
bool session _ register (mixed name [, mixed name 1, ...]) The function accepts one or more variable names as parameters (names are given in brackets, without the $ sign on the left), register them in the currently running session and returns true if registration was successful. Re-writing one variable in a session will not result in an error.

```
<?
session_start ();
session_register ("count");
$ count = @ $ count + 1;
?>
<h2> Counter </h2>
In the current session of work with the browser, you opened this page
<? = $ count?> times (s). Close your browser to reset the counter.
</body>
```

Session group name

It should be noted that on the same site there may be several scripts at once that need PHP session support services. They "know nothing" about each other, so temporary storage for sessions should be selected not only based on user ID, but also on the basis of which of the scripts requested session maintenance.
For clarity, consider an example:

Suppose developer A has written a counter script. It uses the $ count variable and has no problem. Until developer B, who knows nothing about scenario A, created a statistics system that also uses sessions. The worst thing is that it also registers the $ count variable, not knowing that it is already in use. As a result, as always, the user suffers: having started first the developer's script B, and then - A, he sees that the data of the counters are mixed.

We need to somehow distinguish between sessions belonging to one script from sessions belonging to another. Fortunately, the PHP developers envisioned this state of affairs. We can give *session groups* non-overlapping names, and a script that knows the name of its session group will be able to access it. Now developers A and B can protect their scripts from problems with the intersection of variable names. It is enough to tell PHP in the first program that we want to use a group with a name, for example, sesA, and in the second - sesB.

session_name
This function sets or returns the name of the session group that PHP will use to store the registered variables.

Syntax:
string session_name ([string $ newname])

If $ newname is not specified, the current name is returned. If this parameter is specified, the group name

will be changed to $ newname, and the function will return the previous name.

Note that session_name () only changes the name of the current group and session, but does not create a new session and temporary storage. This means that in most cases we must call session_name (group_name) even before its initialization - calling **session_start ()** , otherwise we will get something completely different from what we expected.

If session_name () was not called upon initialization, PHP will use the default name **PHPSESID** .
Example:

```
<?
session _ name (" CounterScript "
session _ start ();
session _ register (" count ");
$ count = @ $ count + 1;
?>
In the current session, you have opened this page <? = $ Count?> Times.
```

Session ID

So, the session ID is the name of the temporary storage that will be used to store session data between script runs. One SID - one store. No SID, no storage or vice versa.
so how do ID and group name compare? The name is just a collective name for several sessions (that is, for many SIDs) started by different users. The same client will never have two different SIDs within the same group name. But his browser can handle multiple SIDs, logically located in different "namespaces".

So, all SIDs are unique and uniquely identify the session on the computer running the script - regardless of the session name. The name specifies the namespace in which the sessions launched by different users will be grouped. One client can have multiple active namespaces at once (that is, multiple session group names).

session_id
This function returns the current session SID.

Syntax:

```
string session_id ([string $ sid])
```

If the $ sid parameter is specified, then the identifier for the active session is changed to $ sid.
By calling session_id () before **session_start ()** , we can connect to any (including someone else's) session on the server, if we know its identifier. We can also create a session with an identifier we want, while automatically setting it in the user's Cookies.

Other functions

session_is_registered
Checks whether a variable is registered or not.

Syntax:

```
bool session_is_registered (string $ name)
```

The function returns true if the variable with the name $ name was registered in the session, otherwise it returns false.

session_unregister
Unregisters a variable.

Syntax:

```
bool session_unregister (string $ name)
```

This function unregisters the $ name variable for the current session. Otherwise, when the script ends, it

looks like a variable named $ name has never been registered.
Returns true if everything went well, false otherwise.
Note that after calling the session_unregister () function, the global variable that has been "invalidated" is not destroyed, but retains its value.

session_unset
Unregisters and destroys global variables.

Syntax:
void session_unset ()

This function, unlike **session_unregister ()** , not only unregisters variables (all session variables, not just one), but also destroys global variables that were registered in the session.

session_save_path
The name of the directory where the session data files will be stored.

Syntax:
string session_save_path ([string $ path])

This function returns the name of the directory in which the files will be placed - temporary storage of session data. If the parameter is specified, the active directory name will be reset to $ path. The function will then return the previous directory.

Overview of handlers

handler_open
This handler should take care of all the work of opening the database for the session group with the name that was passed to it in the parameters.

Syntax :
bool handler_open ($ save_path: string,: string $ session_name)

function is called , when call **session_start ()** . The handler should take over all the work associated with opening the database for the session group named $ session_name. The $ save_path parameter is passed what was specified when calling **session_save_path ()** or the path to the default session data storage files.

handler_close
This handler is called when session data has been written to temporary storage and needs to be closed.

Syntax :
bool handler _ close ()

handler_read
Reading session data.

Syntax:
string handler_read (string $ sid)

This handler is called when you want to read session data with identifier $ sid from temporary storage.
The returned data is presented in the following form:

name1 = value1; name2 = value2; name3 = value3; ...

nameN specifies the name of the next variable registered in the session, and valueN is the result of calling the Serialize () function for the value of this variable.
For example, our record might look like this:

foo | i: 1; count | i: 10;

It says that two integer variables were read from the temporary storage, the first of which is 1, and the second is 10.

handler_write
Writing session data.

Syntax:
string handler_write (string $ sid, string $ data)

This handler is designed to write session data with identifier $ sid to temporary storage - for example, opened earlier by **handler_open ()** handler . The $ data parameter is specified in exactly the same format. In fact, most of the time this function works by writing the $ data string to the database without any changes to it.

handler_gc
Cleans up temporary data storage after a certain period of time.

Syntax:
bool handler_gc (int $ maxlifetime)

This handler is called every time the script exits . If the user finally left the server, it will be charged, the session data in the temporary storage can be destroyed. This is **what the handler_gc ()** function should do . It is passed in parameters that time (in seconds) after which PHP decides to delete all unnecessary data.

session_set_save_handler
Register handlers . When describing handlers, we indicated their names with the handler prefix. In fact, this is not necessary at all. On the contrary, you can name your handlers as you like. But the question arises: how, then, will PHP find them? This is why there is a function for registering handlers, which tells the interpreter which function it should call when an event occurs. **Syntax:** void session_set_save_handler ($ open, $ close, $ read, $ write, $ destroy, $ gc) This function registers subroutines whose names are passed in its parameters as current session handlers. The $ open parameter contains the name of the function that will be called when the session is initialized, and $ close contains the function called when it is closed. In $ read and $ write, you need to specify the names of the handlers, respectively, for reading and writing to temporary storage. The function with the name specified in $ destroy will be called when the session is destroyed. Finally, the handler specified by the $ gc parameter is used as a garbage handler. This function can only be called before the session is initialized, otherwise it is simply ignored.

About sessions and cookies

Problem: - Cookies are disabled
There is a widespread belief that a session without cookies cannot exist. Indeed, Cookies most simply solves the problem of user identification, which is necessary for linking temporary storage and session data. But what to do if the user has disabled the acceptance of cookies in his settings?
In this case, the PHP developers took care of transferring session identifiers not in Cookies, but in some similar way, for example, through the address bar of the browser.

Solution: - changing hyperlinks and forms
In PHP there is one special constant named SID. It always contains the name of the session group and its identifier in the format *name = identifier* . It is in this format that data is received when it comes from the browser's cookies. Thus, we just need to pass the SID constant value to the script so that it "thinks" that the data came from Cookies.
Here's an example of using sessions without cookies:
<?
session_name ("testses");
session_start ();
session_register ("i");
$ i = @ $ i +1;
?>
< body >
You have opened this page
<? = $ i?> times. The counter will reset to zero when the browser is closed.
<A href=sesclick.php?<?=SID?>> Click to record in the counter! </A>
</body>
This example will work if the user actually has cookies disabled. If enabled, PHP will simply not generate the SID constant and will use Cookies.

But in the above method there is one inconvenience, namely, everywhere you need to insert <? = SID?> Into code sections, and if you missed it somewhere, the program may not work!
Fortunately, the PHP developers took this opportunity into account and decided to save us from it. Therefore, if you miss <? = SID?> In any hyperlink, PHP will insert it automatically. In this case, without damaging the rest of the parameters that may be present in the URL.

You can use the following example to check:
<? session _ start ()?>
< body >
< A the href = " ? Php . ? Php "> the PHP </ A >
<A href="php.php?ss=1"> PHP Expressions </A>
Here's what you get when you hover the mouse over these links:

http://www.spravkaweb.ru/php.php?PHPSESSID=81456f6a886f2104
http://www.spravkaweb.ru/php.php?ss=1&PHPSESSID=34f5d04a35601510f45

PHP exists Another possibility to use sessions with disabled Cookies is to add hidden fields to forms that generate a script to pass the session ID to the invoked document.
Here's an example that reveals this possibility:
<? session _ start ()?>
< form action = act . php mathod = post >
</form>
And here's what happens when we view our page as HTML:
<form action = "act.php" method = "post">
<INPUT TYPE = HIDDEN NAME = "PHPSESSID" VALUE = "0a561093f84d4321">
</form>
From the example we can see that PHP has added a hidden field to the form with the required name and value.

Working with WWW

Setting response headers

Header
Displays the header.

Syntax:
int Header (string $ string)

Usually the Header () function is one of the first commands in a script. It is designed to set the response headers that will be passed to the browser - one header per call. Header () must be called before any output statement in the script, otherwise you will receive a warning. Text outside <? and?> are also treated as an output operator.

Example:

```
// redirects the browser to the PHP
Header site ("Location: http://www.php.net");
// now we forcibly terminate the script, since there is nothing else to do after
// redirection exit;
```

Getting request headers

getallheaders
Get all request headers.

Syntax:
array GetAllHeaders ()

The GetAllHeaders () function returns an associative array containing the HTTP headers of the client request that triggered the script. The array keys contain the names of the headers, and the values contain their values.

```
$ headers = GetAllHeaders ();
foreach ($ headers as $ header => $ value)
echo "$ header: $ value <br> \ n";
```

The GetAllHeaders () function is only supported by PHP if it is installed as an Apache module. Otherwise, this function simply will not exist (and it cannot be, because a normal CGI script does not have access to the request headers). In particular, in PHP for Windows (which is most often implemented as a script), the GetAllHeaders () function is not available.

Working with Cookies

A bit of theory A

cookie is a named piece of information that can be stored directly in the user's browser settings between sessions. The reason why cookies are used is a large number of visitors to your server, as well as the unwillingness to have something like a database to store information about each visitor. The search in such a database can be very, very long, and, at the same time, there is no point in centrally storing such fragmentary information. The use of cookies actually shifts the task onto the shoulders of the browser, solving in one fell swoop both the performance problem and the problem of a large database of user information.

The most common way of using cookies is the username and password of a user who uses some of the protected resources of your site. This data is, of course, stored in Cookies between page openings, so that the user does not have to manually re-enter them each time.

Each cookie has a certain lifetime, after which it is automatically destroyed. There are also Cookies that only "live" during the current browsing session.

Each cookie is set by a script on the server. To do this, he must send the browser a special header of the form:

Set-cookie: data

Scripts from other servers, as well as located in another directory, will not be notified about "foreign"

Cookies. For them, they don't seem to exist. And this is correct from a security point of view - who knows how secret the information stored in Cookies can be?

Getting a Cookie

Suppose the script ran and set some kind of Cookie, for example, with the name Cook and the value Val. The next time this script is run (in fact, and all other scripts located on the same server in the same directory or lower in the tree), it will receive a pair of Cook = Val type (via a special environment variable). PHP will intercept this event and automatically create the $ Cook variable with the Val value. That is, the interpreter acts in exactly the same way as if the value of our Cookie came from somewhere in the form. The variable that we set last time will be available now.

setcookie
Sets the cookie.

Syntax:
int setcookie (string $ name [, string $ value] [, int $ expire] [, string $ path] [, string $ domain] [, book $ secure])
Since Cookie is actually a header, you can set it just before the first command to output to the script.

The setcookie () call defines a new Cookie, which is immediately sent to the browser along with the rest of the headers. All arguments other than the name are optional. If only the $ name (Cookie name) parameter is specified, then the Cookie with the specified name is deleted from the user. You can omit arguments you do not want to supply with blank lines "". The arguments $ expire and $ secure cannot be represented by strings, so 0 should be used instead of empty strings.
The $ expire parameter specifies a timestamp, which, for example, can be generated by the time () or mktime () functions.
The $ secure parameter indicates that the Cookie value can only be transmitted over a secure HTTPS connection.

Examples:

```
// Cookie for one session, i.e. before closing the browser
SetCookie ("TextCookie", "value");

// These Cookies are destroyed by the browser 1 hour after setting
SetCookie ("TextCookie", $ val, time () + 3600);
SetCookie ("TextCookie", $ val, time () + 3600, "/ ~ rasmus /", ". Utoronto.ca", 1);
```

After calling SetCookie (), the newly created Cookie immediately appears among the global variables as a variable with the name specified in the $ name parameter. It will appear the next time the script is run - even if SetCookie () is not called in it.

SSI and the virtual () function

A bit of theory

You cannot use two "handlers" for the same document in Apache. In other words, the principle is valid: either PHP or SSI. However, PHP has certain means for "emulating" the SSI-construct *include virtual* .

The *include virtual* construct loads the file whose URL is specified in its parameters, processes it with the necessary handler and displays it in the browser. That is, everything happens as if the specified URL was requested virtual browser Most SSI files are limited to using this feature.

virtual
Simulate include virtual . **Syntax:** int virtual (string $ url) The virtual ( ) function is a procedure that can only be supported if PHP is installed as an Apache module. It does the same as the SSI statement <-

#include virtual = ... -> . In other words, it generates a new request to the server, processes it normally, and then writes the data to standard output. The virtual () function is most often used to run external CGI scripts written in another programming language, or to process SSI files of a more complex structure. In case the script is run, it must generate the correct HTTP headers, otherwise an error message will be displayed. The virtual () function cannot be used to include regular PHP files with code sections - this is done by the **include** statement .

Output control

Introduction

This group of functions allows you to control how PHP outputs information when a script is executed. This can be useful in a variety of situations, especially when sending HTML headers to the browser after the script has started rendering HTML text. (Normally, it is not possible to send a header after text output has started.)
These functions do not affect headers sent by header () or setcookie (), but only functions like echo () and HTML text between PHP blocks. -code.

```
<? php

ob_start ();
echo "Hello \ n"

setcookie ("cookiename", "cookiedata");

ob_end_flush ();

?>
```

In the example above, the output from the echo () command will be stored in the output buffer until the call to ob_end_flush (). At the same time, calling setcookie () successfully saves the cookie without throwing an error.

Output control functions

ob_start
Enable output buffering.

Syntax:
void ob_start ([string output_callback])

After calling this function, output buffering is turned on and, while it is active, none of the output data will be sent to the browser, but will be stored in the internal PHP buffer.

The contents of the buffer can be copied to a string variable using ob_get_contents (). The ob_end_flush () function is used to display the contents from the buffer. The ob_end_clean () function allows you to delete the contents of the buffer.

In the *output_callback* argument, *you* can specify a function that will be automatically called when displaying the contents of the buffer. This is usually used to modify the contents of the buffer before outputting (eg compression). Then, when the ob_end_flush () function is called, the contents of the buffer will be passed to the specified function, and what it returns will be output (note that the function itself

should not output anything).

Buffering can be nested, in which case it is handled appropriately for nesting; and the content output from the lower-level buffer will be included in the upper-level buffer. Remember to call ob_end_flush () as many times as ob_start () was called to display all buffered content.

```
<? php
function c ($ str) {// get the contents of the buffer
  return nl2br ($ str); // returns the contents of the buffer
}
function d ($ str) {// get the contents of the buffer
  return strtoupper ($ str); // returns the contents of the buffer
}
?>
<? php
ob _ start (" c ");
?>

There is a different text ...

<? php
// convert text later to uppercase
ob _ start (" d ");
?>

something else...

<? php
ob_end_flush ();
?>

.....

<? php
ob_end_flush ();
?>
```

ob_get_contents
Get the contents of the output buffer.

Syntax:
string ob_get_contents ()

If buffering is inactive, false is returned.

ob_get_length
Get the length of the data in the output buffer.

Syntax:
string ob_get_length ()

If buffering is inactive, false is returned.

ob_end_flush
Display the contents of the buffer.

Syntax:
void ob_end_flush (void) The

current level buffer is flushed after output, so call ob_get_contents () beforehand if you need to get its contents.

flush
Displays the entire contents of the buffer.

Syntax:
void flush (void);

This function only affects PHP buffering and cannot control the buffering scheme of the web server or browser.

Some servers, especially under Win32, buffer the output of the script before the script ends and the data is sent to the browser.

The browser, in turn, can also buffer the received data before displaying it. Netscape, for example, buffers text until it receives a line terminator or start tag, and for tables, until it receives the </table> tag of the top-level table.

ob_end_clean
Clearing the buffer . **Syntax:** void ob_end_clean (void); The function call disables buffering at the current level.

ob_implicit_flush
Set the buffering mode.

Syntax:
void ob_implicit_flush ([int flag]);

If a nonzero value is specified in the argument, or it is not specified, then the flush () function will be implicitly called during each output operation.

It should be noted that this function often works in a funny way; for example, if you call the ob_end_clean () function at the end of the script, the script will not output anything if the output from the buffer was not explicitly done by other functions.

Controlling PHP Script Execution

Script control functions

set_time_limit
Sets the time limit for script execution.

Syntax:
void set_time_limit (int seconds)

When a script is run, PHP starts a system timer, and if the time (allotted to the script to execute) expires and the script has not finished yet, PHP forces the script to terminate (generating a fatal execution error). This will prevent the accumulation of a large number of scripts that consume server resources, but seem to "hang" (for example, if they find an infinite loop or they are trying to wait for a connection to an unresponsive server).

By default, the allowed execution time of the script is set in the configuration file by the

max_execution_time parameter (usually it is 30 s). But for the current scenario, this time can be changed by calling this function, specifying the time in seconds in its argument. If the value 0 is specified, then the time limit is removed.

The countdown starts from the moment the function is called. For example, if the script has already been executed for 15 seconds, and then the **set_time_limit (20)** function is called , then the total maximum script execution time becomes 35 seconds.

If the script is executed in safe mode (with the safe mode parameter set), then the call to this function is ignored and the value from the configuration file is used.

sleep
Delay in script execution.

Syntax:
void sleep (int seconds);

The **sleep ()** function delays the execution of the script in seconds.

usleep
Script execution delay in microseconds.

Syntax:
void usleep (int micro_seconds);

Delay of script execution in microseconds (micro_seconds).
This feature does not work on Windows.

die
Print a message and end the current script.

Syntax:
void die (string message);

This function displays a message and terminates the execution of the current script. It does not return a value .

```
<? php
$ filename = '/ path / to / data-file';
$ file = fopen ($ filename, 'r')
or die "unable to open file ($ filename)";
?>
```

exit
Exits the current script.

Syntax:
void exit (void);

This function ends the current script. Returns no value.

assert
Check if a value is true.

Syntax:
int assert (string | bool assertion);

The function argument can be a value or a string containing PHP code (as in the eval () function). The function checks if a value (or expression) is false, and if so, performs certain actions.

The behavior of the function is determined by the settings in the configuration file or when calling the

assert_options () function.

Usually this function is used solely for debugging purposes, to check those values that should always be true (for example: module connection, free disk space, etc.).

In general, the execution of the script should not depend on such checks, but use the usual checks of the values returned by the functions.

```
<? php
function handler () {
  echo "\ n * Failed * \ n";
}

assert ("\ $ a = '1';");
echo "a: $ a \ n";
assert (0);
// end script
echo assert_options (ASSERT_BAIL, 1);
// call the handler
assert_options (ASSERT_CALLBACK, "handler");
// don't print PHP messages
@assert (- $ a);
// this line will not be executed
echo "\ n ... \ n"
```

The above example will output : a: 1 Warning: Assertion failed in file.php on line 20 0 * Failed *

assert_options
Define assert options.

Syntax:
mixed assert_options (int parameter [, mixed value])

This function allows you to define the behavior of the assert () construct. Returns the previous value for the parameter (or false on error) specified in the first argument of one of the following constants:

Parameter	ini parameter	Default	Description
ASSERT_ACTIVE	asser.active	1	Allow code to be specified in assert ().
ASSERT_WARNING	assert.warning	1	Issue a PHP warning.
ASSERT_BAIL	assert.bail	0	Terminate execution if not true.
ASSERT_QUIET_EVAL	assert.quiet_eval	0	Do not send messages.
ASSERT_CALLBACK	assert_callback	(null)	Set the function as a handler for "false" assert ().

If a value needs to be overridden, it is specified in the second argument.

eval
Execute a line containing PHP code.

Syntax:
void eval (string code_str);

The **eval ()** function executes the line specified in *code_str* containing PHP code. By the way, this can come in handy for saving the code in a database text field for later execution. Do not forget that the code specified in the line must be syntactically correct (for example, there must be semicolons after each command, etc.), otherwise the script will end with an error on this line. Also keep in mind that the variable values that are

set on this line will be used in the rest of the script.
When changing variable values in eval (), those variables will be changed in the main data as well.

If a **return statement** is specified in the line , then the execution of the specified code will be terminated ahead of schedule and the returned value can be obtained as the value returned by the function itself.

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

The result of executing this code will be : This is a $ string with my $ name in it. This is a cup with my coffee in it.

Connection status

Internally PHP has three connection statuses:

- 0 - NORMAL;
- 1 - ABORTED (interrupted by the user);
- 2 - TIMEOUT (response timeout expired).

When the script is executed normally, the NORMAL state is active. If the user presses the STOP button while loading the page, the ABORTED state becomes active. If the script runs longer than its allotted time, the TIMEOUT status flag is set. It is possible to determine how the script should behave depending on these conditions.

If you want the script to continue its execution when the user disconnects the connection, you need to set the value of the ignore_user_abort = 1 parameter in the configuration file (this can also be done in the Apache configuration files). You can also use the ignore_user_abort () function. Otherwise, the script ends.

To ignore timer completion of the script, use the set_time_limit () function.

If register_shutdown_function () has been set to a "run on script termination" function, then, regardless of the connection status, it will be executed before the script exits. And in the "final" function it will be possible to find out (using the function: connection_aborted (), connection_timeout () and connection_status ()) whether the script was completed normally or ahead of schedule.

connection_aborted
Defines the user to disconnect the connection.

Syntax:
int connection_aborted (void);

Function **connection_aborted ()** returns true, if the connection was aborted by the user.

connection_status Connection
status definitions.

Syntax:
int connection_status (void);

Returns the value of a bit field that allows you to find out in the "final" function, whether the script was terminated early and the reason for this. For example, if 3 (ABORTED | TIMEOUT) is returned, it means that the execution timed out, and that the user refused to load the page.

If it returns 0 (that is, NORMAL), then this means that the script has not been interrupted.

connection_timeout Timeout
definitions.

Syntax:
int connection_timeout (void);

Returns true if the script timed out.

ignore_user_abort Abort the
script on disconnection.

Syntax:
int ignore_user_abort ([int setting]);

The *setting* argument can be used to specify whether the script should be terminated early if communication with the client is lost. If no argument is specified, the current setting is returned.

register_shutdown_function
Sets the function to be executed on shutdown.

Syntax:
int register_shutdown_function (string func);

Registers a function named *func* as a function to run when the script ends.

Note that since no output is available after the function ends, this makes normal debugging tools such as print or echo unavailable for the function registered as "terminator".

Additional functions

get_browser
Define browser capabilities.

Syntax:
object get_browser ([string user_agent]);

The information returned is retrieved from the browscap.ini file. The browser is identified by the value of the $ HTTP_USER_AGENT variable or the value from the *user_agent* argument .

The information is returned in the form of object properties and reflects the capabilities of the client browser (for example, version, whether it supports javascript or cookies).

```
<? php
function list_array ($ array) {
  while (list ($ key, $ val) == each ($ array)) {
    $ str. = "<b> $ key: </b> $ val <br> \ n";
  }
  return $ str;
}

echo "$ HTTP_USER_AGENT <hr>";
$ bouser = get_browser ();
echo list_array ((array) $ browser);
?>
```

Possible output content:

```
Mozilla /4.5 [ en ] ( X 11: Linux 2.2.9 i 586) < hr >
<b> browser_name_pattern: </b> Mozilla / 4 \ .5. * <br>
<b> parent: </b> Netscape <br>
<b> platform: </b> Unknown <br>
<b> majorver: </b> 4 <br>
<b> minorver: </b> 5 <br>
<b> browser: </b> Netscape <br>
<b> version: </b> 4 <br>
<b> frames: </b> 1 <br>
<b> tables: </b> 1 <br>
<b> cookies: </b> 1 <br>
<b> backgroundsounds: </b> <br>
<b> vbscript: </b> <br>
<b> javascript: </b> 1 <br>
<b> javaapplets: </b> 1 <br>
<b> activexcontrols: </b> <br>
<b> beta: </b> <br>
<b> crawler: </b> <br>
<b> authenticodeupdate: </b> <br>
< B > msn : </ b > < br >
```

In order for the function to function, the location of the browscan.ini file must be correctly specified in the configuration file.

highlight_file

Output the content of a color-coded file.

Syntax:

boolean highlight_file (string filename);

The name or path of the file is specified in the argument. Syntax highlight colors are defined in the PHP configuration file. Returns true or false on error.

For example, to force the Apache server, when it receives a request from a URL containing a value of the form "http: //server.name/source/path/to/file.php", displays a listing of the file "http: //server.name/source/path / to/file.php ", do the following.

- Add the following snippet to the httpd.conf file:

```
# Use the "ForceType" directive to specify
# that the source value in the URL is not a directory, but the name of the PHP script
<Location / source>
  ForceType application / x-httpd-php
</Location>
```

- Create the following file named *source* in the web root directory :

```
<HTML> <HEAD>
<TITLE> Source Display </TITLE>
</HEAD>
<BODY bgcolor = # FFEEDD>
<? php
$ script = getenv ("PATH_TRANSLATED");
if (! $ script) {
  echo "<BR> <B> ERROR: Please enter script name </B> <BR>";
} else {
  if (ereg ("(\. php | \ .inc) $", $ script)) {
    echo "<H!> File Listing : $ PATH_INFO </H1> \ n <hr> \ n";
    if (! @ highlight _ file ($ script ))
      echo "File output error";
  } else {
    echo "<H1> ERROR: Only PHP file listings are shown </H1>";
  }
}
```

```
echo "<HR> Printed :" .date ("Y / M / d H: i: s", time ());
?>
</BODY>
</HTML>
```

highlight_string
Color **highlighting** of a string.

Syntax:
void highlight_string (string str);

This function acts like **highlight_file ()** , but it uses the specified string instead of the file content.

show_source
Synonym for highlight_file function . **Syntax :** boolean show_source (string str);

pack
Packing data into a binary string.

Syntax:
string pack (string format [, mixed $ args, ...]);

The **pack ()** function packs the given arguments into a binary string, which is then returned. The format of the parameters, as well as their number, is specified using the *$ format* string , which is a set of one-letter formatting specifiers similar to those specified in **sprintf ()** , but without the% sign. Each specifier can be followed by a number that indicates how much information will be processed by this specifier. Namely, for the formats a, A, h and H, the number specifies how many characters will be placed in a binary string from those that are in the next string parameter when calling the function (that is, it determines the size of the field for outputting the string). In the case of @, it defines the absolute position at which the following data will be placed. For all other specifiers, the following numbers specify the number of arguments that this format applies to. Instead of a number, you can specify *, in this case, it is assumed that the specifier operates on all remaining data.

Here is a complete list of format specifiers:

- a - string, empty spaces in the field are filled with a character with code 0;
- A - string, empty spaces are filled with spaces;
- h - hexadecimal string, least significant bits at the beginning;
- H - hexadecimal string, high order bits at the beginning;
- c - signed byte (symbol);
- C - unsigned byte;
- s - signed short integer (16 bit byte order is determined by the processor architecture);
- S - unsigned short number;
- n - unsigned integer (16 bits, most significant bits at the end);
- v - unsigned integer (16 bits, least significant bits at the end);
- i is a signed integer (the size and byte order is determined by the architecture);
- I - unsigned integer;
- l - signed long integer (32 bits, the order of the characters is determined by the architecture);
- L - unsigned long integer;
- N - unsigned long integer (32 bits, most significant bits at the end);
- V - unsigned integer (32 bits, least significant bits at the end);
- f is a floating point number (depends on the architecture);
- d - double precision floating point number (depending on architecture);
- x - a character with a zero code;
- X - go back 1 byte;
- @ - filling with a zero code to the specified absolute position.

```
// Whole, whole, everything else is sivol
```

```
$ bindata = pack (" nvc *", 0 x 1234, 0 x 5678, 65, 66);
```
After executing the above code, the *$ bindata* line will contain 6 bytes in the following sequence:
0x12, 0x34, 0x78, 0x56, 0x41, 0x42 (in hexadecimal notation).

unpack
Unpacks data from a binary string.

Syntax:
array unpack (string format, string data);

Unpacks data from a binary string into an array according to the format. Returns an array containing the unpacked elements.
```
$ array = unpack (" c 2 chars / nint ", $ binarydata );
```
The resulting array will contain "chars1", "chars2" and "int".

iptcparse
Parses an IPTC binary block for single tags.

Syntax:
array iptcparse (string iptcblock);

This function parses an IPTC binary block for single tags. Returns an array using tagmarker as index and value as value. Returns false on error or if no IPTC data was found.

leak
Simulate a memory leak.

Syntax:
void leak (int bytes);

leak () cuts off a certain amount of memory.
This is useful when debugging a memory manager that automatically cleans up pruned memory when a query is executed.

Memory unit size in bytes argument *bytes* .

serialize
Generates a human-readable representation of the value.

Syntax:
string serialize (mixed value);

serialize () returns a stream of bytes when represented as value, which can be stored somewhere.
This is useful for storing or passing PHP values without losing their type and structure.
Example:
```
// $ session_data contains a multidimensional array
       with session information
// current user. We use
       serialize () to save
// this in the database at the end of the request.

$ conn = odbc_connect ("webdb", "php", "chicken");
$ stmt = odbc_prepare ($ conn,
  "UPDATE sessions SET data =? WHERE id =?");
$ sqldata = array (serialize ($ session_data),
  $ PHP_AUTH_USER);
if (! odbc_execute ($ stmt, & $ sqldata)) {
  $ stmt = odbc_prepare ($ conn,
    INSERT INTO sessions (id, data) VALUES (?,?) ");
  if (! odbc_execute ($ stmt, & $ sqldata)) {
```

```
    / * Something was done wrong. * /
  }
}
```

unserialize
Creates a PHP value from a stored representation.

Syntax:
mixed unserialize (string str);

unserialize () takes one stored value and converts it back to PHP value. Returns the converted value, and can be of type integer, double, string, array, or object. If object was converted, the methods will not be restored.
Example:

```
// Here we use unserialize () to load
      session data from the database
// in $ the session _ data . This example complements
      described in place
// with serialize ().
$ conn = odbc_connect ("webdb", "php", "chicken");
$ stmt = odbc_prepare ($ conn,
  "SELECT data FROM sessions WHERE id =?");
$ sqldata = array ($ PHP_AUTH_USER);
if (! odbc_execute ($ stmt, & $ sqldata) ||
  ! odbc_fetch_into ($ stmt, & $ tmp)) {
// If startup or fetch fails,
// then initialize the array
    $ session_data = array ();
} else {
    // We must have a view at $ tmp [0].
    $ session_data = unserialize ($ tmp [0]);
    if (! is_array ($ session_data)) {
// Something is wrong, initialize the array
        $ session _ data = array ();
    }
}
```

uniqid
Generates a unique identifier.

Syntax:
int uniqid (string prefix [, boolean lcg]);

The **uniqid ()** function returns a unique identifier based on the current time in microseconds and prefixed with *prefix* .
The prefix can be useful, for example, if you are generating identifiers at the same time on separate hosts, which might happen to generate an identifier in the same microsecond. The prefix can be up to 114 characters long.
If an empty string is passed as its value, then the length of the generated identifier will be 13 characters (with lcg = true - 23 characters).

If the optional *lcg* argument is *specified* with the value true, the "combined LCG entropy hash" will be *appended* to the end of the identifier, making its value more unique.

It is also customary to process the received value using cryptographic methods (for example, this is often done in session identifiers).

```
// no random part
$ token = md5 (uniqid (""));
// harder
$ better_token = md5 (uniqid (rnad ()));
```

These strings generate 32 bytes (a 128-bit hexadecimal number): they have as much unnaturalness as you can possibly need.

Mail functions

mail
Sends mail.

Syntax : mail (*$ to, $ subject, $ msg [, $ headers]*);
The **mail ()** function sends a message with the body *$ msg* (it can be a "multi-line string", that is, a variable containing several lines separated by a newline character) to *$ to* . You can specify multiple recipients at once by separating their addresses with spaces in the *$ to* parameter .

Example:
```
mail ("spravka_web@chat.ru spravka_web@hut.ru",
    "My message",
    "First line \ nSecond line \ nThird line"
);
```
If the fourth parameter is specified, the string passed in it is inserted between the end of standard mail headers (such as *To* , *Content-type* , etc.) and the beginning of the message body. Typically this parameter is used to set additional headers for the message.

Example:
```
mail ("spravka_web@chat.ru spravka_web@hut.ru",
    "Topic",
    "Letter body"
    " From : webmaster @ chat . Ru \ n ".
    "Reply-To: webmaster@chat.ru \ n".
    "X-Mailer: PHP /". phpvarsion ()
);
```

Program launch functions

escapeshellcmd Strip
shell wildcards.

Syntax:
string escapeshellcmd (string command);

Strips away any characters in a string that can be used as arbitrary commands in the shell. This function should be used to make sure that all your data is entered correctly, and it is best to insert this function in the exec () or system () function. The standard usage for this function is:
system (EscapeShellCmd ($ cmd))

exec
Run an external program.

Syntax:
string exec (string command [, string array [, int return_var]]);

The **exec ()** function hidden from the user runs the program from the command line, all standard output is disabled. Returns the last line of the program execution result.
If the array parameter is set, then the specified array will be filled with the output from the program. Remember, if the array already contains data, then **exec ()** appends its data to the end of the array. The **unset ()** function can be used to clear an array .
If the *return_var* parameter *is* set along with the array parameter, then the result of the command is written to it.
```
<? php
```

```
$ se = "dir c: \\";
$ s0 = exec ($ se, $ sa, $ sr);
echo "When I ran the command" $ se ", the last line printed was: \ n",
            $ s0, "\ n Return code ($ sr) \ nAnd this is all that was displayed:";
print _ r ($ sa );
?>
```

If you want to run a program in the background (for a long time), then its output stream must be redirected to a file (or another output stream); otherwise, after the valid time of the script execution (waiting for the completion of the external program) has expired, it will be forcibly terminated with an error.

system
Runs an external program with the output.

Syntax:
string system (string command, int [return_var]);

this is the function to run command and print the result. If the second parameter is used, the result of the command is written to it.
The System () call also tries to automatically insert into the web server's output buffer after each line of output if PHP is running as a server model.

passthru
Runs an external program and outputs data directly.

Syntax:
string passthru (string command [, int return_var]);

The **passthru ()** function is similar to the **exec ()** function for running *command* . If return_var is set, then the result of the Unix command is placed here. This function should be used instead of **exec ()** or **system ()** when the output from a Unix command is binary data that should be piped directly back to the browser. This can be used, for example, to run the pbmplus utility to output the image stream directly. By setting the type image / gif and calling the pbmplus program to display the gif image, you can create PHP scripts that display the images directly.

Dynamic loading functions

dl
Load PHP extension library at runtime.

Syntax :
int dl (string library);

```
dl ("extensions / php_db.dll");
```

Loads the PHP extension defined in library.

get_loaded_extensions
Definitions of the list of loaded modules.

Syntax:
array get_loaded_extensions (void);

Returns an array containing a list of PHP module names that were compiled, loaded at PHP startup, and loaded at runtime with the dl () function.

```
print_r (get_loaded_extensions ());
```

Displays information similar to the following:

```
Array
(
    [0] => standard
    [1] => bcmath
    [2] => calendar
```

```
    [3] => ctype
    [4] => com
    [5] => ftp
    [6] => mysql
    [7] => odbc
    [8] => overload
    [9] => pcre
    [10] => session
    [11] => tokenizer
    [12] => xml
    [13] => wddx
    [14] => zlib
    [15] => exif
    [16] => gd
    [17] => zip
)
```

extension_loaded
Checks module loading.

Syntax:
bool extension_loaded (string name);

Returns true if the specified module name has already been loaded. Pay attention to how the module name is spelled and the case of characters.

get_extension_funcs
Definition of module functions.

Syntax:
array get_extension_funcs (string module_name);

Returns an array containing an enumeration of the function names contained in the module module_name. This module must be preloaded.

```
print _ r ( get _ extension _ funcs (" xml "));
```

Information functions

phpinfo
Prints the current state of all PHP options.

Syntax:
int phpinfo ([int what])

To reduce the amount of information displayed, you can specify one of the following what sections (if it is not specified, INFO_ALL is assumed):

- INFO_GENERAL
- INFO_CREDITS
- INFO_CONFIGURATION
- INFO_MODULES
- INFO_ENVIRONMENT
- INFO_VARIABLES
- INFO_LICENSE
- INFO_ALL

This function, which, in general, should not appear in a finished program, displays a large amount of various information in the browser regarding PHP settings and script invocation parameters. Namely, the following is printed to the standard output stream (that is, the user's browser):

- PHP version;

- options that were set when PHP was compiled;
- information about additional modules;
- environment variables, including those set by the server when receiving a request from the user to call the script;
- operating system version;
- the state of the main and local settings of the interpreter;
- HTTP headers;
- PHP license.

The **phpinfo ()** function is mainly used during the initial installation of PHP to test its functionality (it gives a lot of information).
You can check the operation of this function by clicking this link .

phpversion
Returns the current PHP version.

Syntax:
string phpversion ();

Returns a string containing the version name of the PHP interpreter.
```
echo phpversion ();
```
Here's what it should look like:
```
4.3.6
```

phpcredits
HTML listing of PHP developers.

Syntax:
void phpcredits (inf flag);

Displays information about the creators and their contributions to the development of the PHP package.
```
phpcredits ( CREDITS _ GENERAL );
```
Flags can be combined as follows:
```
phpcredits (CREDITS_GROUP + CREDITS_DOCS + CREDITS_FULLPAGE);
```
Below is a list of the available flags:

- CREDITS_ALL - Complete HTML listing .
- CREDITS_DOCS - List of documentation developers.
- CREDITS_FULLPAGE - Usually used in combination with other flags. Selects an option prepared for printing.
- CREDITS_GENERAL - General development of PHP 4.0 and SAPI language
- CREDITS_GROUP - List of kernel developers.
- CREDITS_MODULES - List of add-on modules and their authors.
- CREDITS_SAPI - List of PHP server API module developers.

php_sapi_name
Get the type of interface between the web server and PHP.

Syntax:
string php_sapi_name ();

Returns a string containing the interface type in lowercase. For CGI PHP, this will be the string "cgi", for mod_php under Apache - "apache", etc.
```
$ sapi_type = php_sapi_name ();
if ($ sapi_type == "cgi")
  echo " This is CGI PHP \ n";
else
  echo " This is not CGI PHP but $ sapi_type";
```
Here's what we get for our case:
```
This is not CGI PHP but cgi-fcgi
```

php_uname
Operating system definition.

Syntax:
string php_uname ();

Returns a string containing the name of the operating system, for example "Windows NT MYCOMP 5.1 build 2600".

```
if (substr (php_uname (), 0.7)! = "Windows") {
  die ("This script must run on Windows.");
}
```

ini_set
Change configuration parameter.

Syntax:
string ini_set (string varname, string newvalue);

Sets the specified varname to newvalue. Returns the previous value on success, false on error.

ini_alter
Same as ini_set ().

Syntax :
string ini_alter (string varname, string newvalue);

ini_get
This function gets the values of the configuration parameters.

Syntax:
string ini_get (string varname);

Returns the current value of the configuration parameter specified in the varname variable.
This function allows you to get all the parameters available in PHP.
Returns false on error.

ini_restore Restores the
configuration **parameter** .

Syntax:
string ini_restore (string varname);

Sets the varname configuration parameter to its original value.

```
echo ini_set ("precision", 20) .ini_get ("precision").
     ini_restore ("precision"). ini_get ("precision");
// Prints 14 20 14
```

get_cfg_var
Retrieves parameter values directly from php.ini file.

Syntax:
string get_cfg_var (string varname);

It should be noted that unlike the ini_get () function, which returns the current parameter value, the get_cfg_var () function returns the parameter value, which is set in the php.ini configuration file. Also, this function does not return other parameters (for example, from the server configuration itself).

getenv
The function returns the value of an environment variable.

Syntax :
string getenv (string varname);

```
$ ip = getenv ("REMOTE_ADDR");
echo " Your IP address is $ ip";
```

Here's what you get as a result of the work:

```
Your IP address: 127.0.0.1
```

The list of environment variables can be viewed in Applications -> Environment Variables , or using the phpinfo () function .
This feature does not work in PHP ISAPI module.

putenv
Sets an environment variable.

Syntax:
void putenv (string setting);

```
putenv ("UNIQID = $ uniquid");
```

get_magic_quotes_gpc
Gets the current value of the magic_quotes_gpc parameter.

Syntax:
long get_magic_quotes_gpc ();

This function will return 0 for Off and 1 for On.

get_magic_quotes_runtime
Used to get the current value of the magic_quotes_runtime patameter.

Syntax:
long get_magic_quotes_runtime ();

This function will return 0 for Off and 1 for On.

set_magic_quotes_runtime
Designed to set the current value of the magic_quotes_runtime patameter.

Syntax:
long set_magic_quotes_runtime (int new_setting);

To set magic_quotes_runtime to Off, set new_setting to 0, and to set On to 1.

php_logo_guid
Function for getting the PHP logo GUID.

Syntax :
string php_logo_guid ();

Line

```
echo php_logo_guid ();
```

will return

```
PHPE9568F34-D428-11d2-A769-00AA001ACF42
```

zend_logo_guid
Function to get the GUID of the Zend logo.

Syntax :
string zend_logo_guid ();

Line
```
echo zend_logo_guid ();
```
will return
```
PHPE9568F35-D428-11d2-A769-00AA001ACF42
```
1

Part 5. Interaction with databases

MySQL database

Working with databases

mysql_connect
Establishes a network connection to a MySQL database.

Syntax:
int mysql_connect ([string $ hostname [: port] [: / path / to / socket] [, [, string $ username [, string $ password]]])

The *mysql_connect ()* function establishes a network connection to the MySQL database, located on the host *$ hostname* , and returns the identifier of the open connection. All further work is carried out with this identifier. When registering, the username is *$ username* and the password is *$ password* . The *$ hostname* string can also include the port number in the form "hostname: port" or the path to the socket for the local machine on Unix systems - ": / path / to / socket" (if the MySQL server is configured not for standard, but for some then another port).
If an error occurs, a warning is issued. You can block the error message by prefixing the function name with the "@" operator.
The next time the function is run with the same arguments, the second connection will not be opened, and the function will return the identifier of the existing one.
At the end of the script, it is usually customary to close the connections with the **mysql_close ()** function , but you do not need to do this, since PHP automatically closes all (volatile) connections when the script ends.

```
<? php
  $ conn = mysql_connect ("localhost", "username", "pass")
      or die ("Connection not established!");
  print ("Connection established!");
  mysql _ close ($ conn );
?>
```

mysql_pconnect
Establishes a stable network connection to the MySQL database.

Syntax:
int mysql_pconnect ([string $ hostname [: port] [: / path / to / socket] [, [, string $ username [, string $ password]]])

The **mysql_pconnect ()** function establishes a stable network connection to the MySQL database located on host *$ hostname* and returns the identifier of the open connection. All further work is carried out with this identifier. When registering, the username is *$ username* and the password is *$ password* . The *$ hostname* string can also include the port number in the form "hostname: port" or the path to the socket for the local machine on Unix systems - ": / path / to / socket" (if the MySQL server is configured not for standard, but for some then another port).
On error, a warning is issued. You can block the issue of an error message by preceding the function name with the "@" operator.
The next time the function is run with the same arguments, the second connection will not be opened, and the function will return the identifier of the existing one.
mysql_pconnect () behaves similarly to **mysql_connect** , with two differences:
- Before connecting, the function tries to check if there is already an open connection. If so, an identifier is returned instead of creating a new connection.
- When the script ends, the connection is not closed, but remains valid for further use, ie. function **mysql_close ()** can not close the connection created with **mysql_pconnect ()** .

mysql_close
Closes a previously established database connection.

Syntax:
int mysql_close ([int link_identifier])

Closes the connection to the MySQL server with *link_identifier* , or the last open connection if used without an identifier.
Returns true on success, or false on error.
Using this function is optional, since PHP automatically closes all unstable connections when the script exits.
Connections established by **mysql_pconnect ()** are not closed.

```
<? php
  $ conn = mysql _ connect (" localhost ", " username ", " pass ")
      or die ("Connection not established!");
  print ("Connection established!");
  mysql _ close ($ conn );
?>
```

mysql_change_user
Changes connection parameters.

Syntax:
int mysql_change_user (string user, string password [, string database [, int link_identifier]])

If no database or connection is specified, then the last active database is used.
If authorization has not occurred, the connection parameters are not changed.
Works with MySQL 3.23.3 and up.

mysql_list_dbs
Returns the list of databases on the server.

Syntax:
int mysql_list_dbs ([int link_identifier])

Returns a recordset containing a list of databases on the server.

```
$ bd = mysql_connect ("localhost", "name", "pass");
$ bd_list = mysql_list_dbs ($ bd);
while ($ row = mysql_fetch_object ($ bd_list)) {
  echo $ row -> Database . "\ n ";
}
```

It should be noted that the list of databases can be obtained without privileges, i.e. without specifying the access password.

mysql_db_name
Returns the name of the database from the list.

Syntax:
int mysql_db_name (int result, int row [, mixed field])

The *result* parameter specifies a handle to the recordset retrieved using **mysql_list_dbs ()** . The *row* argument specifies the record number.
This function returns false on error.

```
mysql_connect ("localhost", "username", "pass");
$ db_list = mysql_list ();
for ($ i = 0; $ i <($ cnt = mysql_num_rows ($ db_list)); $ i ++) {
  echo mysql_db_name ($ db_list, $ i). "\ n";
}
```

The function was previously called **mysql_dbname ()** .

mysql_select_db

Select one MySQL database.

Syntax:
int mysql_select_db (string database_name [, int link_identifier])

Returns true on successful close, or false on error.
If you plan to open only one connection to the database for the entire duration of the script, you may not save the returned value, and also omit the identifier when calling all other functions.
Before sending the first query to the MySQL server, you need to specify which database we are going to work with. This is what this function is for. It notifies that further operations on the *link_identifier* connection (or on the last open connection if the specified parameter is not specified) will use the database_name *database* .
If at the time of calling this function there are no connections to the database, then the **mysql_connect ()** function is called indirectly with default parameters.

mysql_create_db
Creates a MySQL database.

Syntax:
int mysql_create_db (string dbname [, int link_identifier])

This function creates a new MySQL database named *dbname* using the *link_identifier* connection .

```
$ db = mysql_connect ("localhost", "name", "pass");
if (mysql_create_db ("my_db_name")) {
  echo " my_db_name database has been created ");
} else {
  echo " Error creating database :% s \ n" .mysql_error ());
}
```

mysql_drop_db
Dropping the MySQL database.

Syntax:
int mysql_drop_db (string database_name [, int link_identifier])

The **mysql_drop_db ()** function **drops** the database_name *database* available on the *link_identifier* connection .
Returns true on successful deletion, false on error.

mysql_list_tables
Returns a list of tables in the database.

Syntax:
int mysql_list_tables (string database [, int link_identifier])

The function returns the result identifier (one column), which contains the names of all tables present in the database. To retrieve these names, you can use the **mysql_result ()** function with a column number of 0, or the **mysql_tablename ()** function .

The following example will list all the names of the databases and tables they contain:

```
$ db = mysql_connect ("localhost", "user_name", "");
$ db_list = mysql_list_dbs ($ db);
while ($ r_db = mysql_fetch_object ($ db_list)) {
  echo $ r_db-> Database. "\ n";
    // print the list of tables
  $ t_list = mysql_list_tables ($ r_db-> Database);
  for ($ i = 0; $ i <mysql_num_rows ($ t_list); $ i ++) {
    echo "-" .mysql_tablename ($ t_list, $ i). "\ n";
  }
}
```

mysql_tablename
Returns the name of the table in the database.

Syntax:
int mysql_tablename (int result, int i)

The function returns the name of the table with the number *i* from the set of records obtained using the **mysql_list_tables ()** function .

```
$ db = mysql_connect ("localhost", "user_name", "");
$ result = mysql_list_tables ("db_name");
$ i = 0;
while ($ i <mysql_num_rows ($ result)) {
  $ t_name [$ i] = mysql_tablename ($ result, $ i);
  echo $ t_name [$ i]. "<BR>";
  $ i ++;
}
```

mysql_query
Sends a query to the MySQL database.

Syntax:
int mysql_query (string query [, int link_identifier])

This function sends a query *query to* the database associated with the *link_identifier*. If no identifier is specified, the last open connection is taken into account. If a connection has not been established before, then the **mysql_connect ()** operation is performed with the default parameters.
The SQL expression specified in the *query* parameter must not end with ";".
If the expression contains errors, or its execution results in errors, then the **mysql_query ()** function returns false.
A successful query returns a recordset that can be processed by the following functions:

- mysql_result () - get an element of a recordset
- mysql_fetch_array () - add a record to the array
- mysql_fetch_row () - write a record to a numbered array
- mysql_fetch_assoc () - add a record to the associative array
- mysql_fetch_object () - Fetch a record into an object

To find out how many records were found by a SELECT command, use the **mysql_num_rows ()** function .
Use **mysql_affected_rows ()** to find out how many records have changed as a result of DELETE, INSERT, REPLACE, or UPDATE **queries** .

After processing the query results, it can be removed with the **mysql_free_result ()** function . But this is not necessary, since the results are themselves destroyed after the script finishes.

mysql_db_query
Sends a query to the specified MySQL database.

Syntax:
int mysql_db_query (string database, string query [, int link_identifier])

This function is equivalent to the following sequence of functions:

```
mysql_select_db (string database [, int link_identifier]);
mysql_query (string query [, int link_identifier]);
```

mysql_num_rows
Returns the number of rows in a query result.

Syntax:
int mysql_num_rows (int result)

This function returns the number of records found as a result of executing an SQL SELECT statement (database search).

```
<?
$ link = mysql_connect ("localhost", "username", "password");
mysql_select_db ("database", $ link);

$ result = mysql_query ("SELECT * FROM table1", $ link);
$ num_rows = mysql_num_rows ($ result);

echo "Received lines: $ num_rows \ n";
?>
```

mysql_affected_rows
Returns the number of changed records in the MySQL database.

Syntax:
int mysql_affected_rows ([int link_identifier]);

Function **mysql_affected_rows ()** returns the number of records that have been changed in the database as a result of DELETE, INSERT query, REPLACE, or UPDATE.
If the last query was a DELETE command without a WHERE constraint (i.e. all records were deleted from the table), then our function will return 0.

mysql_insert_id
Gets the inserted identifier.

Syntax:
int mysql_insert_id ([int $ link_identifier])

The function returns, immediately before calling it, the generated record ID for the auto-incrementing field after executing the insert command. It is reasonable to call it only immediately after the insert statement is executed, for example, in the following context:

```
mysql_query ("insert into Table (field 1, field 2) values  (" aa "," bb ")");
$ id = mysql_insert_id ();

The record you just inserted can now be

referenced using the $ id: $ r = mysql_query ("select * from Table where id = $ id");
$ Row = mysql_fetch_array ($ r);
```

mysql_data_seek
Sets the current row pointer.

Syntax:
int mysql_data_seek (int result, int row_number)

This function sets the current row pointer in *result* to the *row_number* position , so that the next call to **mysql_fetch_row ()** and **mysql_fetch_array ()** will return the field values of that particular row.
Records are numbered starting from 0.
Returns false in case of an error or if the lines ran out.

mysql_free_result
Destroys a recordset.

Syntax:
int mysql_free_result (int result)

This function frees the memory occupied by the *result* recordset returned by the query.

This function is necessary when you need to save memory, because PHP automatically frees memory when the script ends.

Processing query results

mysql_result
Get a specific result field.

Syntax:
int mysql_result (int result, int row [, mixed field])

The function returns the value of the field *field* in the result row with *row* number . The *field* parameter can specify not only the name of the field, but also its number - the position at which the column "stood" when creating a table, as well as the full name of a field of the form: "table_name.field_name". However, it is recommended that field names are used wherever possible.

The function is universal: it can be used to "bypass" the entire result one by one. And although it is not forbidden, it is not recommended to do it, because **mysql_result ()** is rather slow.

mysql_fetch_array
Retrieves the next record from the result and places it into an associative array.

Syntax:
array mysql_fetch_array (int result [, int result_type])

The **mysql_fetch_array (**) function returns the next row of the result in the form of an associative array, where each field is associated with an element with a key that matches the field name. Additionally, the array contains elements with numeric keys and values corresponding to the values of the fields with these indices. In the returned array, they are placed immediately after the elements with "normal" keys.

The *result_type* parameter specifies the type of the returned array and can take one of the following values: MYSQL_NUM, MYSQL_ASSOC, MYSQL_BOTH (by default).
The question may arise: why do we need numeric indices at all. The answer is simple: the fact is that as a result of the selection, in fact, there may be fields (in fact, columns) with the same names, but, accordingly, with different indices. This happens when a selection in a SELECT is made simultaneously from several tables.

```
mysql_connect ($ host, $ user, $ pass);
$ result = mysql_db_query ("database", "select id, name from tabl");
while ($ row = mysql_fetch_array ($ result)) {
  echo "id:". $ row ["id"]. "<BR>";
  echo "id:". $ row [0]. "<BR>";
  echo "name:". $ row ["name"]. "<BR>";
  echo "name:". $ row [1]. "<BR>";
};
mysql_free_result ($ result);
```

mysql_fetch_row
Writes a record to a numbered array.

Syntax:
array mysql_fetch_row (int result)

The function returns an array-list with the values of the fields of the next row of the result *result* . If the pointer to the current position of the result was positioned past the last record (that is, the rows ran out), false is returned. The current position is shifted to the next record, so the next call to **mysql_fetch_row ()** will return the next row in the result.
Each field in the record is stored in a numbered array element. Numbering starts from 0.

```
$ r = mysql_query ("select * from OutTable where age <30");
while ($ Row = mysql_fetch_row ($ r)) {
```

```
  // process the $ Row
}
```
As you can see, the loop will end as soon as the lines end, i.e. when **mysql_fetch_row ()** will return false.

mysql_fetch_object
Get a record in the properties of an object.

Syntax:
object mysql_fetch_object (int result)

The function returns an object whose properties contain the fields of the current record. Returns false if there are no more entries.
```
mysql _ connect ($ host , $ user , $ pass );
$ result = mysql_db_query ("database", "select * from table");
while ($ rows = mysql_fetch_object ($ result)) {
  echo $ rows-> id;
  echo $ rows-> name;
};
```

mysql_fetch_lengths
Returns the length of the item in the record.

Syntax:
array mysql_fetch_lengths (int result)

The **mysql_fetch_lengths (**) function returns the length of the value retrieved by **mysql_fetch_row ()** , **mysql_fetch_array (),** or **mysql_fetch_object ()** .

For example , in the following example :
```
$ arr = mysql_fetch_row ($ result);
$ len = mysql_fetch_lengths ($ result);
```
the $ len array will contain the length of the corresponding elements of the $ arr array, i.e. $ len [0] = strlen (arr [0]) etc.

mysql_fetch_field
Returns information about object properties and record field.

Syntax:
object mysql_fetch_field (int result [, int field_offset])

The optional *field_offset* parameter specifies the number of the field whose properties we want to get. If this parameter is not specified, each call to **mysql_fetch_field ()** returns the properties of the next field in the *result* recordset .

The returned object has the following properties:
- name - field name
- table - the name of the table to which the field belongs
- max_length - maximum field length
- not_null - 1 if the field is allowed to be empty
- primary_key - 1 if the field is key
- unique_key - 1 if only unique values are allowed in the field
- multiple_key - 1, if it is allowed to have duplicate values in the field
- numeric - 1 if the field is numeric
- blob - 1, if the field is BLOB
- type - field type
- unsigned - 1 if the field is numeric unsigned
- zerofill - 1, if the field is filled with zeros

```
mysql _ connect ($ host , $ user , $ pass );
$ result = mysql_db_query ("database", select * from table ");
for ($ i = 0; $ i <mysql_num_fields ($ result); $ i ++) {
```

```
  echo "Properties of $ i field: <BR>";
  $ param = mysql_fetch_field ($ result);
  if (! $ param) echo "No property information!";
  echo "<PRE>
name: $ param-> name
table: $ param-> table
max_length: $ param-> max_length
not_null: $ param-> not_null
primary_key: $ param-> primary_key
unique_key: $ param-> unique_key
multiple_key: $ param-> multiple_key
numeric: $ param-> numeric
blob: $ param-> blob
type: $ param-> type
unsigned: $ param-> unsigned
zerofill: $ param-> zerofill
</PRE> ";
}
```

mysql_field_seek
Moves the cursor to the specified field.

Syntax :
int mysql_field_seek (int result, int field_offset)

This function is redundant . The following snippets will be equivalent :

```
$ param = mysql_fetch_field ($ result, field_offset);
and
mysql_field_seek ($ result, field_offset);
$ param = mysql_fetch_field ($ result);
```

mysql_field_name
Returns the name of the field.

Syntax:
string mysql_field_name (int result, int filed_index)

The **mysql_field_name (**) function returns the name of the field that is located in *result* with index *filed_index* (numbering starts at 0).

```
$ result = mysql_query ("SELECT id, name from table");
echo mysql_field_name ($ result, 1); // Output : name
```

mysql_field_table
Returns the name of the table from which the field was retrieved.

Syntax:
string mysql_field_table (int result, int field_offset)

Returns the name of the table from which the field was retrieved at offset *field_offset* as a result of *result* .

mysql_field_len
Returns the length of the field.

Syntax:
int mysql_field_len (int result, int filed_offset)

The function returns the length of the field from *result* . The field, as usual, is specified by specifying its offset. The length here does not mean the size of the field data in bytes, but the size that was specified when it was created. For example, if a field is of type *varchar* and was created (along with the table) with type *varchar (100)* , then 100 will be returned for it.

mysql_field_type
Returns the type of the resulting recordset.

Syntax:
string mysql_field_type (int result, int filed_offset)

This function is similar to **mysql_filed_name ()** , only it returns not the name, but the type of the corresponding column in the result. They can be, for example, int, double, real, etc.

```
mysql_connect ($ host, $ user, $ pass);
mysql_select_db ("mydb");
$ result = mysql_query ("SELECT * FROM tabl");
$ fields = mysql_num_fields ($ result);
$ rows = mysql_num_rows ($ result);
$ i = 0;
$ table = mysql_field_table ($ result, $ i);
echo " Table " $ table " has a $ fields fields and $ rows records <BR>";
echo " Table structure : <BR>";
while ($ i <$ fields) {
  $ type = mysql_field_type ($ result, $ i);
  $ name = mysql_field_name ($ result, $ i);
  $ len = mysql_field_len ($ result, $ i);
  $ flags = mysql_field_flags ($ result, $ i);
  echo $ type. "". $ name. "". $ len. "". $ flags. "<BR>";
  $ i ++;
}
```

mysql_field_flags
This function returns the flags that were used to create the specified field in the table.

Syntax:
string mysql_field_flags (int result, int field_offset) The

returned string is a set of words separated by spaces, so you can convert it to an array using the **explode ()** function :

```
$ Flags = explode ("", mysql_field_flags ($ r, $ field_offset));
```

Field records in MySQL can have the following properties - flags : "not_nul", "primary_key", "unique_key", "multiple_key", "the blob", "an unsigned", "zerofill", "binary", "an enum", "auto_increment" , "timestamp".

mysql_list_fields
Returns a list of table fields.

Syntax:
int mysql_list_fields (string dbname, string tblname [, int link_identifier])

The **mysql_list_fields (**) function returns information about the specified *tblname* table in the *bdname* database using the *link_identifier* , if specified (otherwise, the last open connection). The return value is the identifier of the result, which can be parsed by conventional means. In case of an error, -1 is returned, the text of the error message can be received in the usual way.

```
$ link = mysql_connect ($ host, $ user, $ pass);
$ fields = mysql_list_fields ("db1", "table", $ link);
$ colums = mysql_num_fields ($ fields); // number of fields in the table
// Next, print out the names of all fields in the table
for ($ i = 0; $ i <$ colums; $ i ++) {
  echo mysql_field_name ($ fields, $ i). "<BR>";
}
```

mysql_num_fields
This function returns the number of fields in one row of the result, i.e. the number of columns in the result.

Syntax:
int mysql_num_fields (int result)

By virtue of the above, the function allows to determine the horizontal dimension of the "two-dimensional result array".

mysql_errno
Returns the last error number.

Syntax:
int mysql_errno ([int link_identifier])

This function returns the number of the last logged error, or 0 if there are no errors.
The *link_identifier* can be omitted if only one connection was established during the script run.

```
mysql_connect ("dbname");
echo mysql_errno (). ":" .mysql_error (). "<BR>";
```

mysql_error
Returns an error message.

Syntax:
string mysql_error ([int link_identifier])

This function returns a string containing the text of the error message, or an empty string if there were no errors.

```
mysql_connect ("dbname");
echo mysql_errno (). ":" .mysql_error (). "<BR>";
```

1

Part 6. Graphics

Image manipulation and the GD library

Image options

Application and Installation

Here we will look at the idea of creating pictures with a script on the fly. This can be very useful when creating counter scenarios, graphs, header pictures, and much more.

For this kind of activity, there is a special library called GD. It contains many functions (such as drawing lines, stretching / compressing an image, filling to the border, displaying text, etc.) that can be used by programs that support this library.

GD support is included when PHP is compiled and installed. Some hosting providers may not have it.

To connect the module on your local disk, open the *php.ini* file in Notepad from the directory with Windows files (usually C: \ Windows)
Then :
1. Configure the following parameter :
extension_dir = C: \ Program Files \ PHP4 \ extensions
  This is where we notify PHP that it should look for modules
  in the C: \ Program Files \ PHP4 \ extensions directory .
2. Find the commented out line *; extension = php_gd.dll*
   and uncomment it i.e. remove; at the beginning.
3. Save changes in *php.ini file*

imageTypes
Determines the image formats supported by PHP.

Syntax:
int imageTypes (void)

The function returns a bitmask of graphic formats supported by this version of the GD library: IMG_GIF | IMG_JPG | IMG_PNG | IMG_WBMP

```
<? php
if (imageTypes () && IMG_PNG) echo "PNG is supported";
?>
```

GetImageSize
Determines the size of the picture.

Syntax:
array GetImageSize (string filename [, array imageinfo])

This function is used to quickly determine the size (in pixels) and format of the image in the script, the file name of which is passed to it in the first parameter. It returns a list of four elements. The first element (with key 0) stores the width of the image in pixels, the second (with key 1) stores its height. The array cell with key 2 is determined by the image format: 1 if it is GIF, 2 if it is JPG, 3 if it is PNG, and 4 if it is SWF. The next element with key 3 will contain, after calling the function, a line similar to the following: height = sx width = sy, where sx and sy are the image width and height, respectively. This application is intended to make it easier to insert image size data into an <*img*> tag that can be generated by a script:

```
<? php
$ size_img = GetImageSize ("img / image.jpg");
echo "<IMG src = 'img / image.jpg' $ size_img>";
?>
```

If a second optional imageinfo array was specified when calling the function, additional information about the file can be written to it. This could be, for example, various JPG APP markers (embedded information). The iptcparse () function allows you to convert this data into a readable form:

```
<? php
$ size _ img = GetImageSize (" img / image . jpg ", & $ info _ arr );
if (isset ($ info_Arr ["APP13"])) {
  $ iptc = iptcparse ($ info_arr ["APP13"]);
  var _ dump ($ iptc );
};
?>
```

This function does not require the GD library.

imageSX
Determines the width of the image.

Syntax:
int imageSX (int im)

The function returns the horizontal size of the image specified by its identifier *im* , in pixels.

imageSY
Determines the height of the picture.

Syntax:
int imageSY (int im)

The function returns the vertical size of the image specified by its identifier *im* , in pixels.

read_exif_data
Read EXIF headers from a JPEG file.

Syntax :
array read_exif_data (string filename)

The *filename* parameter cannot be a URL.

The function returns an associative array in which the indexes are the names of the EXIF headers.
EXIF headers usually store digital camera information (in various forms).

```
<? php
$ exif = read_exif_data ("img / file.jpg");
print _ r ($ exif );
?>
```

This example will output something like:

```
Array
(
   [FileName] => file.jpg
   [FileDateTime] => 1064566998
   [FileSize] => 31646
   [CameraMake] => Eastman Kodak Company
   [CameraModel] => KODAK DC265 ZOOM DIGITAL CAMERA (V01.00)
   [DateTime] => 2002: 08: 31 02:12:45
   [Height] => 454
   [Width] => 620
   [IsColor] => 1
   [FlashUsed] => 0
   [FocalLength] => 8.0mm
   [RawFocalLength] => 8
   [ExposureTime] => 0.004 s (1/250)
   [RawExposureTime] => 0.0040000001899898
   [ApertureFNumber] => f / 9.5
   [RawApertureFNumber] => 9.5100002288818
   [FocusDistance] => 16.66m
   [RawFocusDistance] => 16.659999847412
   [Orientation] => 1
   [ExifVersion] => 0200
)
```

This function is available , if the connected library EXIF. For this you must either remove the comment

from the line ; extension = php_exif.dll in file windows \ php.ini (must have extension = php_exif.dll), or compile PHP with the option --enable-exif.

The GD library is not required for this function to work.

imageInterlace
Sets the interlacing .

Syntax :
int imageInterlace (int im [, int interlace])

If the second optional parameter *interlace is* specified in the function , and it is equal to 1, then the picture *im is* displayed interlaced, if equal to 0, then sequentially.

The function returns the current interlacing setting.

gd_info
Returns information about the GD library.

Syntax :
array gd_info (void)

The function returns an array containing the version and parameters of the installed GD library.

```
<? php
$ gd = gd_info ();
echo "<pre>";
print_r ($ gd);
echo "</pre>";
?>
```

The above example will output something like the following:

```
Array
(
    [GD Version] => bundled (2.0.22 compatible)
    [FreeType Support] => 1
    [FreeType Linkage] => with freetype
    [T1Lib Support] =>
    [GIF Read Support] => 1
    [GIF Create Support] =>
    [JPG Support] => 1
    [PNG Support] => 1
    [WBMP Support] => 1
    [XBM Support] => 1
    [JIS-mapped Japanese Font Support] =>
)
```

image_type_to_mime_type
Returns the Mime-Type of the image type .

Syntax :
string image_type_to_mime_type (int imagetype)

The function returns the MIME type of the image, specified by a constant in the *imagetype* parameter .

```
<? php
header ("Content-type:". image_type_to_mime_type (IMAGETYPE_PNG));
?>
```

List of constants and return values of the **image_type_to_mime_type ()** function :

- **IMAGETYPE_GIF** - image / gif
- **IMAGETYPE_JPEG** - image / jpeg
- **IMAGETYPE_PNG** - image / png
- **IMAGETYPE_SWF** - application / x-shockwave-flash
- **IMAGETYPE_PSD** - image / psd
- **IMAGETYPE_BMP** - image / bmp
- **IMAGETYPE_TIFF_II** - image / tiff
- **IMAGETYPE_TIFF_MM** - image / tiff
- **IMAGETYPE_JPC** - application / octet-stream

- **IMAGETYPE_JP2** - image / jp2
- **IMAGETYPE_JPX** - application / octet-stream
- **IMAGETYPE_JB2** - application / octet-stream
- **IMAGETYPE_SWC** - application / x-shockwave-flash
- **IMAGETYPE_IFF** - image / iff
- **IMAGETYPE_WBMP** - image / vnd.wap.wbmp
- **IMAGETYPE_XBM** - image / xbm This function does not require the GD library.

Manipulating images

imageCreate
Creates an empty image.

Syntax:
int imageCreate (int x, int y)

Creates an empty *x* by *y* pixel image and returns its ID. After the picture is created, all work with it is carried out precisely through this identifier, by analogy with how we work with a file through its descriptor.

Example:
Creating a new picture with GD and displaying it in the browser screen:

```
<? php
header ("Content-type: image / png");
$ im = @imagecreate (50, 100)
   or die ("Can't open new picture!");
$ background_color = imagecolorallocate ($ im, 255, 255, 255);
$ text_color = imagecolorallocate ($ im, 233, 14, 91);
imagestring ($ im, 1, 5, 5, "A Simple Text String", $ text_color);
imagepng ($ im);
?>
```

imageCreateFromPng
Creates a picture from a PNG file.

Syntax:
int imageCreateFromPng (string filename)
This function loads images from a PNG file into memory and returns its ID. As after the call to **imageCreate ()** , further work with the image is possible only through this identifier. When loaded from disk, the image is unpacked and stored in memory in an unpacked format, so that you can perform various operations with it as quickly as possible, such as scaling, drawing lines, etc.

Example:
An example of finding an error when opening a graphic file.

```
function LoadPNG ($ imgname ) {
 $ im = @imagecreatefrompng ($ imgname); / * Attempt to open * /
 if (! $ im) {/ * See if it failed * /
   $ im = imagecreate (150, 30); / * Create a blank image * /
   $ bgc = imagecolorallocate ($ im, 255, 255, 255);
   $ tc = imagecolorallocate ($ im, 0, 0, 0);
   imagefilledrectangle ($ im, 0, 0, 150, 30, $ bgc);
   / * Output an errmsg * /
   imagestring ($ im, 1, 5, 5, "Error loading $ imgname", $ tc);
 }
 return $ im;
}
```

imageCreateFromJpeg
Creates a picture from a JPEG file.

Syntax:
int imageCreateFromJpeg (string filename)

This function loads images from a file into memory and returns its ID. As after the call to **imageCreate ()** , further work with the image is possible only through this identifier. When loaded from disk, the image is unpacked and stored in memory in an unpacked format, so that you can perform various operations with it as quickly as possible, such as scaling, drawing lines, etc.

imageCreateFromGif
Creates a picture from a GIF file.

Syntax:
int imageCreateFromGif (string filename)
This function loads images from a file into memory and returns its ID. As after the call to **imageCreate ()** , further work with the image is possible only through this identifier. When loaded from disk, the image is unpacked and stored in memory in an unpacked format, so that you can perform various operations with it as quickly as possible, such as scaling, drawing lines, etc.

It's worth mentioning that GD does not support GIF since version 1.6. Therefore, this function is practically not used.

imagePng
The function outputs the image in PNG-format to any browser or file.

Syntax :
int imagePng (int im [, string filename])

This function saves the image specified by its identifier and located in memory to disk, or outputs it to the browser.

Of course, the image must first be loaded or created using the **imageCreate ()** function , i.e. we need to know its id *im* .

If *filename is* omitted, the compressed data in the appropriate format is output directly to standard output, i.e. into the browser. At the same time, the required *Content-type* header is not displayed, so you need to display it manually using **Header ()** .
In fact, you must invoke one of three commands, depending on the type of the image:

Header ("Content-type: image / png") for PNG.

Example:
An example of using the imagepng () function:

```
<? php
$ im = imagecreatefrompng (" test . png ");
Header ("Content-type: image / png")
imagepng ($ im);
?>
```

imageJpeg
Send the JPEG image to the browser or save it to a file.

Syntax :
int imageJPEG (int im [, string filename [, int quality]])

This function saves the image specified by its ID and resides in memory to disk, or outputs it to the browser.

Of course, the image must first be loaded or created using the **imageCreate ()** function , i.e. we need to know its id *im* .

If *filename is* omitted, the compressed data in the appropriate format is output directly to standard output, i.e. into the browser. At the same time, the required *Content-type* header is not displayed, so you need to display it manually using **Header ()** .
In fact, you have to call one of three commands, depending on the image type:

Header ("Content-type: image / jpeg") for Jpeg

The third optional parameter, *quality*, specifies the image quality (from 0 to 100).

```
<? php
```

```
$ im = imageCreateFromJPEG ("img / file.jpg");
Header ("Content-type: image / jpeg");
imageJPEG ($ im, "", 30);
?>
```

image2WBMP
Output image to browser or file.

Syntax :
int image2WBMP (resource image [, string filename [, int threshold]])

The function outputs the image specified by the *image* descriptor to the browser, or to a file whose name is specified by the optional *filename* parameter .
If the image is output to the browser, you need to set its WBMP type as *image / vnd.wap.wbmp using the* Header () function:

```
<? php
$ file = "php.png";
$ image = imagecreatefrompng ($ file);

header ("Content-type:". image_type_to_mime_type (IMAGETYPE_WBMP));
image2wbmp ($ image); // Output the wbmp image to the browser
?>
```

The **image2WBMP ()** function is available in PHP only if the GD library version is 1.8 or lower.

imageGif
Send the GIF image to the browser or save it to a file.

Syntax :
int imageGIF (int im [, string filename])

The functions save the image specified by its identifier and located in memory to disk, or output it to the browser.

Of course, the image must first be loaded or created using the **imageCreate ()** function , i.e. we need to know its id *im* .

If *filename is* omitted, the compressed data in the appropriate format is output directly to standard output, i.e. into the browser. At the same time, the required *Content-type* header is not displayed, so you need to display it manually using **Header ()** .
In fact, you must invoke one of three commands, depending on the type of image:

Header ("Content-type: image / gif").

Because since GD 1.6 does not support GIF format, this function is rarely used.

imageCopy
Copies part of the picture.

Syntax:
int imageCopy (int dst _ im , int src _ im , int dst _ x , int dst _ y , int src _ x , int src _ y , int src _ w , int src _ h)

Function copies a rectangular region ranging from a position (*src_x*, *src_y*) width *src_w* and height *src_h* from figure *src_im* in Figure *dst_im* , giving copied area offset (*dst_x*, *dst_y*).

The following example will *copy the* whole picture *file1.png* to *file2.png*

```
<? php
// Create the first image based on the finished image
$ im1 = imageCreateFromPNG ("img / file1.png");
// Determine its size
$ size_x = imageSX ($ im1);
$ size _ y = imageSY ($ im 1);
// Create a second empty picture
$ im2 = imageCreate ($ size_x, $ size_y);
// Copy the whole picture from the first image to the second
imageCopy ($ im2, $ im1,0,0,0,0, $ size_x, $ size_y);
```

```
// Save the copied image to a file
imagePNG ($ im 2, " img / file 2. png ");
?>
```

imageCopyResized
Copies part of the picture with scaling.

Syntax :
int imageCopyResized (int dst_im, int src_im, int dstX, int dstY, int srcX, int srcY, int dstW, int dstH, int srcW, int srcH)

This feature is one of the most powerful and versatile. With it, you can copy images (or parts of them), move or scale them.

dst_im sets the image identifier in which the result of the function will be placed. This image must have already been created or uploaded and properly sized.

src_im - the identifier of the image being worked on. However, *src_im* and *dst_im* may be the same.

The *srcX, srcY, srcW, srcH parameters* set the area inside the original image, over which the operation will be performed - respectively, the coordinates of its upper left corner, width and height.

Finally, the four *dstX, dstY, dstW, dstH* sets the place on the *dst_im* image , into which the rectangle indicated in the previous four will be "squeezed". Note that if the width or height of the two rectangles do not match, then the picture will automatically be stretched or shrunk as needed.

In the following example, *file1.jpg* is halved and written to *file2.jpg* :

```
<? php
$ old = imageCreateFromJpeg ("img / file1.jpg");
$ w = imageSX ($ old);
$ h = imageSY ($ old);
$ w_new = rounr ($ w / 2);
$ h_new = round ($ h / 2);
$ new = imageCreate ($ w_new, $ h_new);
imageCopyResized ($ new, $ old, 0, 0, 0, 0, $ w_new, $ h_new, $ w, $ h);
imageJpeg ($ new, "img / file2.jpg");
imageDestroy ($ old);
imageDestroy ($ new);
?>
```

imageDestroy
Destruction pattern .

Syntax :
int imageDestroy (int im)

The function destroys the *im* descriptor of the previously created drawing (like closing the file fclose () after opening fopen ()).

Working with RGB color

imageColorAllocate Creates a new color and **puts**
it in the picture's palette.

Syntax :
int imageColorAllocate (int im, int red, int green, int blue)

The function returns the identifier of the color associated with the corresponding RGB triplet. The first parameter of the function requires the identifier of the image loaded into memory or created before.
The *red* , *green,* and *blue* parameters set the red, green, and blue color components, respectively. The values of these parameters must be in the range from 0 to 255, or from 0x00 to 0xFF.
Almost every color that is planned to be used in the future must be obtained (determined) by calling this function.

Example:
An example of using the imageColorAllocate () function:

```
<? php
... ... ...
// white
$ white = imagecolorallocate ($ im, 255, 255, 255);
$ white = imagecolorallocate ($ im, 0xFF, 0xFF, 0xFF);
// black
$ black = imagecolorallocate ($ im, 0, 0, 0);
$ black = imagecolorallocate ($ im, 0x00, 0x00, 0x00);
... ... ...
?>
```

imageColorDeAllocate Excludes a
color from the picture's palette.

Syntax :
int imageColorDeAllocate (int im, int color)

This feature removes the figure palette *im* color *color* , which has been pre-entered in the drawing function **imagecolorallocate ()** .

Example:
An example of using the imageColorDeAllocate () function:

```
<? php
... ... ...
$ white = imageColorAllocate ($ im, 255, 255, 255);
imageColorDeAllocate ($ im, $ white);
... ... ...
?>
```

imageColorSet
Replaces the color of a specific palette element.

Syntax :
bool imageColorSet (int im, int index, int red, int green, int blue)

This function sets for the element of the palette *index of the* picture *im the* values of the color components: *red* , *green* (green), *blue* (blue). In this case, all parts of the picture filled with this color will also change their shade.

imageColorClosest Gets the closest
color of the palette to the specified.

Syntax :
int imageColorClosest (int im, int red, int green, int blue)

Instead of trying to find empty space in the color picker, this function simply returns the identifier of the color that already exists in the drawing and is closest to the requested one. Thus, no new color is added to the palette. If the palette is not large, then the function may not return exactly the color that you expect. For example, in a palette of three colors "red-green-blue", a request for yellow will most likely return the identifier for green — which, from GD's point of view, is closest to "green."

imageColorTransparent
Specifies the transparency color .

Syntax :
int imageColorTransparent (int im [, int color])

This function tells GD that the corresponding *color* (specified by its ID using the imageColorAllocate () function) in the image *im* (*im* is the image ID specified by the imageCreate () function) should be denoted transparent. Returns the identifier of the previously set transparent color, or false if one was not previously defined.
It should be noted that not all formats support the setting of transparent color - for example, JPEG cannot contain it.

imageColorsForIndex
Getting the RGB components of the palette element.

Syntax :

array imageColorsForIndex (int im, int index)

The function returns an associative array with the keys *red*, *green*, *blue* (in that order), which correspond to values equal to the values of the RGB components in the color identifier *index* . But we can ignore the keys and convert the returned value as a list:

```
<? php
... ... ...
$ color = imageColorAt ($ im, 0,0);
list ($ r, $ g, $ b) = array_values  (imageColorsForIndex ($ im, $ color));
echo "R = $ r, g = $ g, b = $ b";
... ... ...
?>
```

imageColorAt
Returns the color index of the point.

Syntax :
int imageColorAt (int im, int x, int y)

This function returns the color of the point located at coordinates (*x*, *y*).
If PHP is compiled with GD library 2.0 or higher, and the image is truecolor, then this function will return the color identifier, not its RGB representation.

```
<? php
$ im = imageCreateFromPng ("file.png");
$ rgb = ImageColorAt ($ im, 100, 100);
$ r = ($ rgb >> 16) & 0xFF;
$ g = ($ rgb >> 8) & 0xFF;
$ b = $ rgb & 0xFF;
?>
```

imageColorsTotal Gets the
number of colors in the palette.

Syntax :
int imageColorsTotal (int im)

The function returns the number of colors in the palette of the specified image.

imageColorExact
Get the color index of the palette.

Syntax :
int imageColorExact (int im, int red, int green, int blue)

The function returns the index of the specified color (*red, green, blue*) in the *im* image palette .
The function will return -1 if the specified color is not in the image palette.

imageColorResolve Find
or create the specified color.

Syntax :
int imageColorResolve (int im, int red, int green, int blue)

The function returns the index of the specified color (*red, green, blue*) in the *im* image palette .
If there is no such color in the palette, then it is created.

imageGammaCorrect
Applies a gamma correction to the image.

Syntax :
int imageGammaCorrect (int im, double inputgamma, double outputgamma)

This function makes gamma corrections to the image specified by the *im* descriptor .
Parameter *inputgamma* sets the input range and *outputgamma* - range output.

Graphic primitives

imageSetPixel

Draws a pixel .

Syntax :
int imageSetPixel (int im, int x, int y, int color)

Outputs one pixel of *color* in the image *im* located at point (*x, y*).

imageLine
Draws a solid thin line.

Syntax :
int imageLine (int im, int x1, int y1, int x2, int y2, int color)

This function draws a solid thin line in the image *im* passing through the points (*x1, y1*) and (*x2, y2*) with *color* . The line turns out to be loosely connected .

```
<? php

function imagelinethick ($ image, $ x1, $ y1, $ x2, $ y2, $ color, $ thick = 1)
{
   / * this way it works well only for orthogonal lines
   imagesetthickness ($ image, $ thick);
return imageline ($ image, $ x1, $ y1, $ x2, $ y2, $ color);
* /
if ($ thick == 1) {
   return imageline ($ image, $ x1, $ y1, $ x2, $ y2, $ color);
}
$ t = $ thick / 2 - 0.5;
if ($ x1 == $ x2 || $ y1 == $ y2) {
   return imagefilledrectangle ($ image,
    round (min ($ x1, $ x2) - $ t),
    round (min ($ y1, $ y2) - $ t),
    round (max ($ x1, $ x2) + $ t),
    round (max ($ y1, $ y2) + $ t), $ color);
}
$ k = ($ y2 - $ y1) / ($ x2 - $ x1); // y = kx + q
$ a = $ t / sqrt (1 + pow ($ k, 2));
$ points = array (
   round ($ x1 - (1 + $ k) * $ a), round ($ y1 + (1- $ k) * $ a),
   round ($ x1 - (1- $ k) * $ a), round ($ y1 - (1 + $ k) * $ a),
   round ($ x2 + (1 + $ k) * $ a), round ($ y2 - (1- $ k) * $ a),
   round ($ x2 + (1- $ k) * $ a), round ($ y2 + (1 + $ k) * $ a),
);
imagefilledpolygon ($ image, $ points, 4, $ color);
return imagepolygon ($ image, $ points, 4, $ color);
};
?>
```

imageDashedLine
Draws a dashed line .

Syntax :
int imageDashedLine (int im, int x1, int y1, int x2, int y2, int color)

This function works in much the same way as **imageLine ()** , except that it draws a dotted line instead of a solid one. Unfortunately, neither the size nor the step of the strokes can be specified, so if you need a dashed line of an arbitrary texture, you will have to do some mathematical calculations and use **imageLine ()** .

imageRectangle
Draws a rectangle .

Syntax :
int imageRectangle (int im, int x1, int y1, int x2, int y2, int color)

This function draws a rectangle in the image *im* with a border of 1 pixel in *color* .
The upper left corner is specified by (*x1, y1*) and the lower right corner is specified by (*x2, y2*).

imageFilledRectangle

Sketch a rectangular area .

Syntax :
int imageFilledRectangle (int im, int x1, int y1, int x2, int y2, int color)

This function draws a filled rectangle in the image specified by the identifier *im* with the color *color* (obtained, for example, using the **imageColorAllocate ()** function). The coordinates (*x1*, *y1*) and (*x2*, *y2*) set the coordinates of the upper-left and lower-right corners, respectively (the counting, as usual, starts from the upper corner and goes from left to right and from top to bottom).

This function is often used to completely fill a newly created drawing, for example, with a transparent color:

```
<? php
$ im = imageCreate (100,100);
$ color = imageColorAllocate ($ i, 0,0,0);
imageColorTransparent ($ im, $ color);
imageFilledRectangle ($ im, 0,0, imageSX ($ im) -1, imageSY ($ im) -1, $ color);
// then work with an initially transparent background
?>
```

imageArc
Draws a portion of an ellipse .

Syntax :
int imageArc (int im, int cx, int cy, int w, int h, int s, int e, int color)

This function draws in the image *im* an arc of an ellipse sector from angle *s* to *e* (angles are indicated in degrees counterclockwise, measured from the horizontal). The ellipse is drawn large enough to fit within the rectangle (*w*, *h*), where *w* and *h* specify its width and height. *cx* and *cy* are the coordinates of the center of the ellipse. The shape itself is not painted over, only its outline is outlined, for which the color *color is used* .

```
<? php
// create a 200x200 image
$ img = imagecreate (200, 200);
// set the color of the circle
$ white = imagecolorallocate ($ img, 255, 255, 255);
// draw a circle
imagearc ($ img, 100, 100, 150, 150, 0, 360, $ white);
// output the image to the browser
header ("Content-type: image / png");
imagepng ($ img);
// close the picture
imagedestroy ($ img);
?>
```

imageFill
Fill the bounded area with color.

Syntax :
int imageFill (int im, int x, int y, int color)

This function fills a solid area containing a point at coordinates (*x*, *y*) with *color* . You need to measure that modern filling algorithms work quite efficiently, so you shouldn't worry too much about the speed of its work. Only those points will be shaded to which a "one-color strongly connected path" can be drawn from the point *x*, *y* .

Two points are called strongly connected if they have the same at least one coordinate, and in the other coordinate they differ by no more than 1 in any direction.

imageFillToBorder
Fill the area enclosed by the border.

Syntax :
int imageFillToBorder (int im, int x, int y, int border, int color)

This feature is very similar to **ImageFill ()** , only it performs color shading *color* is not monochrome pixels

and all, but until then, until you reach the border color *border* .

imagePolygon
Draws a polygon with the given vertices.

Syntax :
int imagePolygon (int im, array points, int num_points, int color)

This function draws in the image *im a* polygon given by its vertices. The coordinates of the angles are passed in the *points* array , with *$ points [0] = x0, $ points [1] = y0, $ points [2] = x1, $ points [3] = y1* , etc.
The *num_points* parameter specifies the total number of vertices, in case there are more points in the array than you need to draw. The polygon is not painted over - only its border is drawn with the color *color* .

```
<? php
// create a 400x300 image
$ image = imagecreate (400, 300);

// set the border color of the polygon
$ col_poly = imagecolorallocate ($ image, 255, 255, 255);

// draw a polygon
imagepolygon ($ image,
        array (
            0, 0,
            100, 200,
            300, 200
        ),
        3, $ col_poly);

// image output to the browser
header ("Content-type: image / png");
imagepng ($ image);
?>
```

imageFilledPolygon
Draws a filled polygon with the specified vertices.

Syntax :
int imageFilledPolygon (int im, array points, int num_points, int color)

This function does almost the same thing as **imagePolygon ()** , except for one very important property: the resulting polygon is completely filled with *color* .
In this case, the concave parts of the figure are processed correctly, if it is not convex.

```
<? php
// set an array with the coordinates of the corners
$ values  = array (
  0 => 40, // x1
  1 => 50, // y1
  2 => 20, // x2
  3 => 240, // y2
  4 => 60, // x 3
  5 => 60, // y 3
  6 => 240, // x 4
  7 => 20, // y 4
  8 => 50, // x 5
  9 => 40, // y 5
  10 => 10, // x6
  11 => 10, // y6
);

// create a 250x250 image
$ im = imagecreate (250, 250);
```

```
// set the fill color of the polygon
$ blue = imagecolorallocate ($ im, 0, 0, 255);

// draw a polygon
imagefilledpolygon ($ im, $ values, 6, $ blue);

// display the cartoon in the browser and close it
header ('Content-type: image / png');
imagepng ($ im);
imagedestroy ($ im);
?>
```

Working with fixed fonts

The GD library has some features for working with text and fonts. Fonts are special resources that have their own identifier and are most often loaded from a file or embedded in GD. Each character in the font can only be displayed in monochromatic mode, i.e. "drawn" characters are not supported. There are only 5 built-in fonts (identifiers from 1 to 5), most often they include monospaced characters of different sizes. The rest of the fonts must be preloaded.

imageLoadFont Font
loading .

Syntax :
int imageLoadFont (string file)

The function loads the font *file file* and returns the font identifier - this will be a number greater than 5, because the first five numbers are reserved as inline numbers. The file format is binary and therefore depends on the architecture of the machine. This means that the font file must be generated at least on a machine with the same processor architecture as the one on which you intend to use PHP.

Font file format

Bias	A type	Description
Byte 0-3	long	Number of characters in the font (nchars)
byte 4-7	long	The index of the first character of the font (usually 32 - space)
byte 8-11	long	The width (in pixels) of each character (width)
byte 12-15	long	The height (in pixels) of each character (height)
byte 16 -...	array	An array with information about the style of each character, one byte per pixel. Thus, there are width * height * nchars bytes per character. 0 means there is no point in this position, everything else is its presence.

The left column sets the offset of the beginning of the data within the file, and groups of numbers, written with a hyphen, determine to which address the data continues.

imageFontHeight
Sets the font height.

Syntax:
int imageFontHeight (int font)

The function returns the height in pixels of characters in the specified font.

imageFontWidth
Sets the font width.

Syntax:

int imageFontWidth (int font)

The function returns the width in pixels of characters in the specified font.

imageString
Prints a string horizontally.

Syntax :
int imageString (int im, int font, int x, int y, string s, int color)

The function outputs the string *s* to the image *im* using the font *font* and the color *color* .
The (*x*, *y*) coordinates will be the coordinates of the upper left corner of the rectangle the line is inscribed in.

If the *font* parameter is specified as 1, 2, 3, 4, or 5, then the font of the appropriate size is displayed.

```
<? php
 // create a 100x30 image
 $ im = imagecreate (100, 30);

 // set the text color
 $ textcolor = imagecolorallocate ($ im, 0, 0, 255);

 // display the label in the upper left corner
 imagestring ($ im, 5, 0, 0, "Hello world!", $ textcolor);

 // display the image to the browser
 header ("Content-type: image / jpg");
 imagejpeg ($ im);
?>
```

imageStringUp
Prints a string vertically.

Syntax :
int imageStringUp (int im, int font, int x, int y, string s, int color)

This function also prints a line of text, but not horizontally, but vertically.
The upper left corner is specified by coordinates (*x*, *y*).

If the *font* parameter is specified as 1, 2, 3, 4, or 5, then the font of the appropriate size is displayed.

imageChar
Outputs the character horizontally .

Syntax :
int imageChar (int im, int font, int x, int y, string c, int color)

The function displays the character *c* horizontally at the location in the figure specified by the coordinates (*x*, *y*). The font of the character is specified by the *font* parameter . If this parameter takes a value between 1 and 5, then embedded fonts are used. The color of the symbol is set by the *color* parameter .

```
<? php
// create a 100x100 image
$ im = imagecreate (100, 100);

$ string = "PHP";

// set the color of the symbol
$ black = imagecolorallocate ($ im, 0, 0, 0);

// Display the "P" character in the upper left corner
imagechar ($ im , 1, 0, 0, $ string , $ black );

// display the image in the browser
header ("Content-type: image / png");
imagepng ($ im);
?>
```

imageCharUp
Displays the character vertically .

Syntax :
int imageCharUp (int im, int font, int x, int y, string c, int color)

The function displays the character *c* in a vertical position at the location in the figure specified by coordinates (*x*, *y*). The font of the character is specified by the *font* parameter . If this parameter takes a value between 1 and 5, then embedded fonts are used. The color of the symbol is set by the *color* parameter .

Working with TrueType and PostScript Type 1 fonts

The GD library also supports PostScript and TrueType fonts. For the functions below to work, PHP must be compiled and installed with the FreeType library available at http://www.freetype.org . It is installed by default in the Windows version of PHP.

imageTTFText
Draws text with TrueType font.

Syntax :
array imageTTFText (int im, int size, int angle, int x, int y, int color, string fontfile, string text)

This function puts the string *text* in the image *im* with *color* . As usual, *color* must be a valid color identifier. The *angle* parameter sets the angle in degrees of the output line, counted from the horizontal counterclockwise. The (*x*, *y*) coordinates indicate the position of the so-called *base point of the line* (usually its lower-left corner). The *size* parameter specifies the size of the font to be used when displaying the string. *fontfile* must contain the name of the TTF file that contains the font.

The function returns a list of 8 elements. The first pair of them sets the coordinates (x, y) of the upper left corner of the rectangle described around the line of text in the image, the second pair sets the coordinates of the upper right corner, etc. Since in general a string can have any slope *angle* , 4 pairs of coordinates are required here.

The text string *text* can contain UTF-8 character sequences (in the form & # 123;) to output characters with codes greater than 255.

When using negative color values of the index *color* is disconnected font smoothing (antialiasing).

This function requires the GD library and FreeType.

```
<? php
  header ("Content-type: image / jpeg");
  $ im = imagecreate (400, 30);
  $ white = imagecolorallocate ($ im, 255, 255, 255);
  $ black = imagecolorallocate ($ im, 0, 0, 0);

  // Replace path by your own font path
  imagettftext ($ im, 20, 0, 10, 20, $ black, "/path/arial.ttf",
  "Testing ... Omega: & # 937;");
  imagejpeg ($ im);
  imagedestroy ($ im);
?>
```

The following example displays a line in the center of the picture.

```
<? php
$ gi = imageCreate (200,100);
$ bg = imageColorAllocate ($ gi, 0,220,0);
$ tx = imageColorAllocate ($ gi, 25,2,228);
$ w = imageSX ($ gi); // image width
$ h = imageSY ($ gi); // picture height
imageFilledRectangle ($ gi, 0,0, $ w, $ h, $ bg);

$ szf = 20; // font size
$ ang = 240; // line rotation angle
```

```
$ str = "Heyou"; // line text
$ font = " symbol . ttf " // font file
$ sz = imageTTFBBox ($ szf, $ ang, $ font, $ str);
$ sdx = $ sz [4] / 2;
$ sdy = ($ sz [7] + $ sz [3]) / 2;
imageTTFText ($ gi, $ szf, $ ang, $ w / 2- $ sdx, $ h / 2- $ sdy, $ tx, $ font, $ str);
Header ("Content-Type: image / png");
imagePng ($ gi, "file.png");
?>
```

imageTTFBBox
Calculates the area occupied by a TrueType font line.

Syntax :
attay imageTTFBBox (int size, int angle, string fontfile, string text)

This function does not output anything to the image, but simply determines whether the size and position would take a string of text *text* size *size bed* , derived angle *angle* in any drawing. The *fontfile* parameter specifies the absolute path to the font file that will be used for output.

The returned list contains all information about the dimensions of the string in a format similar to that **produced by the imageTTFText ()** function . However, the order of the points in it is different.

Content of the array returned by the imageTTFBBox () function:

0 and 1 - (x, y) lower left corner
2 and 3 - (x, y) lower right corner
4 and 5 - (x, y) upper right corner
6 and 7 - (x, y) upper left corner

Coordinates can have negative values.
The function requires the GD library and FreeType.

imagePSLoadFont
Load from PostScript Type 1 font file.

Syntax :
int imagePSLoadFont (string filename)

Returns a handle to the loaded font, or FALSE on error (also displays a warning).

```
<? php
header ("Content-type: image / jpeg");
$ im = imagecreate (350, 45);
$ black = imagecolorallocate ($ im, 0, 0, 0);
$ white = imagecolorallocate ($ im, 255, 255, 255);
$ font = imagepsloadfont ("bchbi.pfb"); // or locate your .pfb files on your machine
imagepstext ($ im, "Testing ... It worked!", $ font, 32, $ white, $ black, 32, 32);
imagepsfreefont ($ font);
imagejpeg ($ im, "", 100); // for best quality ... your mileage may vary
imagedestroy ($ im);
?>
```

This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePSFreeFont
Unloading font PostScript Type 1.

Syntax :
void imagePSFreeFont (int fontindex)

This function frees memory from the font specified by the *fontindex* parameter .
This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePSEncodeFont
Sets the text encoding scheme.

Syntax :

int imagePSEncodeFont (int font_ndex, string encodingfile)

Loads the file conversion *encodingfile* for font *font_index* . Because PostScript fonts do not use characters with codes greater than 127 by default, transcoding is required if a language other than English is required. The file format is described in the Tllibs documentation, and 2 ready-made files are supplied with the library: IsoLatinl.enc and IsoL.atin2.enc.

If the transcoding is used constantly, set the ps.default_encoding parameter in the configuration file to the name of the transcoding file that will be loaded automatically.

This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePsExtendFont
Scale the font.

Syntax:
bool imagePsExtendFont (int font _ index , float extend)

The function stretches or *shrinks the* font specified by the *font_index* parameter to the size specified by the *extend* parameter .
If the value of the *extend* parameter is less than 1, then the font will be reduced.

This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePsSlantFont Sets the **slant of the**
font.

Syntax:
bool imagePsSlantFont (int font _ index , double slant)

The function sets the slant of the *font_index* to the value specified by the *slant* parameter .

This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePSBBox
Calculates the area occupied by a line of PostScript Type 1 font.

Syntax :
array imagePSBBox (string text, int font, int size [, int space [, int tightness [, float angle]]])

Calculations are made based on the following arguments:
size - font size in pixels;
space - change the size of spaces in relation to normal (can be negative);
tightness - spaces between characters in relation to) to normal (can be negative);
angle - line angle in degrees.

Values *space* and *tightness* are measured in fractions of a space (1/1000).
Arguments *space* , *tightness are* , *angle* is not required.

The calculation results are not accurate enough. The function returns an array:

- 0 - lower left corner, X-coordinate;
- 1 - lower left corner, Y-coordinate;
- 2 - upper right corner, X-coordinate;
- 3 - upper right corner, Y - coordinate.

This feature is only available if PHP was compiled with the --enable-t1lib option.

imagePSText Displays text over an image **using**
PostScript Type 1 font.

Syntax :
array imagePSText (resource image, string text, int font, int size, int foreground, int background, int x, int y [, int space [, int tightness [, float angle [, int antialias_steps]]]])

The *size* parameter sets the size of the font.
The *x* , *y* coordinates indicate the lower left corner of the first character.
The *foreground* and *background* arguments set the colors of the text and background (the background is only needed to smooth the font).
The *antialias_steps* argument allows *you* to specify the number of colors to use when antialiasing the text (valid values are 4 and 16). For fonts less than 20, use a higher value as it improves readability; for large fonts, use a lower value as this will improve performance.
The *angle* parameter sets the slope of the text in degrees.

The function returns an array, similar to **imagepsbbox ()** .

This feature is only available if PHP was compiled with the --enable-t1lib option.

PDF documents

Introduction

PDF functions allow PHP to create PDFs using the PDF library created by Thomas Merzem (http://www.pdflib.com/pdflib/index.html); you might also need the JPEG (ftp://ftp.uu.net/graphics/jpeg/) and TIFF (http://www.libtiff.org/) libraries .

Pdflib comes with good documentation describing the features of the library. Function names and arguments are identical in the library and PHP. Dimensions and coordinates are measured in Postscript units (72 per inch), but this depends on the selected resolution.

The analogue of the library is ClibPDF.

Versions below 3.0 pdflib are not supported in PHP 4.

```
<? php
$ fp = fopen ("test.pdf". "w");
$ pdf = pdf_open ($ fp);
pdf_set_info ($ pdf, "Author", "Uwe Streinmann");
pdf_set_info ($ pdf, "Title", "Test for PHP PDFlib");
pdf_set_info ($ pdf, "Creator", "See Author");
pdf_set_info ($ pdf, "Subject", "Testing");
pdf_begin_page ($ pdf, 595, 842);
pdf_add_outline ($ pdf, "Page 1");
pdf_set_font ("$ pdf," Times-Roman ", 30," host ");
pdf_set_value ($ pdf, "textrendering", 1);
pdf_show_xy ($ pdf, "Times Roman outlined", 50, 750);
pdf_moveto ($ pdf, 50, 740);
pdf_lineto ($ pdf, 330, 740);
pdf_stroke ($ pdf);
pdf_end_page ($ pdf);
pdf_close ($ pdf);
fclose ($ fp);
echo "<A href=getpdf.php> finished </A>";
?>
<? php
// script getpdf.php just returns the pdf document
$ fp = fopen ("test.pdf", "r");
header ("Content-type: application / pdf");
fpassthru ($ fp);
fclose ($ fp);
?>
```

Opening a document

pdf_set_info
Filling in the document information field.

Syntax :
void pdf_set_info (int pdf_document, string fieldname, string value)

Possible *fieldname* fields :

- Subject
- Title
- Creator
- Author
- Keywords
- One , defined by the user . The function must be called before the pages are created.

```
<? php
$ fd = fopen ("test.pdf", "w");
$ pdfdoc = pdf_open ($ fd);
pdf_set_info ($ pdfdoc, "Author", " Author name ");
pdf_set_info ($ pdfdoc, "Creator" , " Name of the creator ");
pdf_set_info ($ pdfdoc, "Title", " Title ");
pdf_set_info ($ pdfdoc, "Subject", " Subject ");
pdf_set_info ($ pdfdoc, "Kewwords" , " Key , the word ");
pdf_set_info ($ pdfdoc, "CustomField", " Something Else ");
pdf_begin_page ($ pdfdoc, 595, 842);
pdf_end_page ($ pdfdoc);
pdf_close ($ pdfdoc);
?>
```

This function replaces a pdf_set_info_keyword (), pdf_set_info_title (), pdf_set_info_subject (), pdf_set_info_creator ().

pdf_open
Opens a new pdf document.

Syntax:
int pdf_open (int file) This

function makes a file opened by fopen () a pdf document. If you do not provide a file descriptor, it is created in memory and can then be written to standard output or sent to the browser. The function returns a handle to the document, which should be specified in subsequent pdf functions.

pdf_close
Closes a pdf document.

Syntax :
void pdf_close (int pdf_document)
pdf_begin_page
Beginning of a new page.

Syntax:
void pdf_begin_page (int pdf_document, double width, double height)

The *height* and *width* arguments specify the height and width of the page. After entering information on the page, it should be closed with the pdf_end_page () function.

pdf_end_page
End of page.

Syntax:
void pdf_end_page (int pdf_document)

After this function, modification of this page is not possible.

Work with text

pdf_show
Displays text at the current position.

Syntax:
void pdf_show (int pdf_document, string text)

The current position and the current font are used for output.

pdf_show_boxed
Displays text in a rectangular area.

Syntax:
void pdf_show_boxed (int pdf_document, string text, double x, double y, double width, double height, string mode [, string feature]) The

lower left corner of the output area is set to (*x* : *y*); height and width - *height* , *width* . The *mode* argument specifies the alignment of the text: if the height and width are zero, then the possible values are:
- left
- right
- center,

if they are not equal to zero, then
- justify
- fulljustify

If *feature* is "blind", no text is displayed.

The function returns the number of characters that do not fit in the specified rectangle.

pdf_show_xy
Displays text at the specified position.

Syntax :
void pdf_show_xy (int pdf_document, string text, double x, double y)

pdf_set_font
Selects the font, size and encoding.

Syntax:
void pdf_set_font (int pdf_document, font_name string, double size, string encoding [, int embed])

Argument encoding type *encoding* may be:
- winansi (default)
- builtin
- host
- macroman , etc. If the last argument is set to 1, the font will be embedded in the pdf document (otherwise not). If the font is common, do not embed it due to the increased size of the document. The function should be called after pdf_begin_page ().

pdf_set_leading
Sets the space between lines of text.

Syntax :
void pdf_set_leading (int pdf_document, double distance)

Used when displaying text with the pdf_continue_text () function .

pdf_set_parameter
Sets the string value of the pdflib parameter.

Syntax :
void pdf_set_parameter (int pdf_document, string name, string value)

pdf_get_parameter
Get the string value of the pdflib parameter.

Syntax :
void pdf_get_parameter (int pdf_document, string name [, double modifier])

The *modifier* argument is used as needed .

pdf_set_value
Sets the numerical value of the pdflib parameter.

Syntax :
void pdf_set_value (int pdf_document, string name, double value)

pdf_get_value
Get the numerical value of the pdflib parameter.

Syntax:
void pdf_get_value (int pdf_document, string name [, double modifier])

The *modifier* argument is used when needed.

pdf_set_text_rendering
Sets the text rendering method.

Syntax :
void pdf_set_text_rendering (int pdf_document, string mode)

Deprecated , use pdf_set_value ().

pdf_set_horiz_scaling
Sets the horizontal scaling of the text.

Syntax :
void pdf_set_horiz_scaling (int pdf_document, double scale)

pdf_set_text_rise
Sets the text rise.

Syntax :
void pdf_set_text_rise (int pdf_document, double rise)

pdf_set_text_matrix
Sets the font transformation matrix.

Syntax:
void pdf_set_text_matrix (int pdf_document, array matrix)

This function is not available since pdflib 2.3.

pdf_set_text_pos
Sets the position of the font.

Syntax:
void pdf_set_text_pos (int pdf_document, double x-coor, double y-coor)

Sets the position of the text output by a subsequent call to pdf_show ().

pdf_set_char_spacing
Sets the spacing between characters.

Syntax :
void pdf_set_char_spacing (int pdf_document, double space)

Deprecated , use pdf_set_value ().

pdf_set_word_spacing
Sets the spacing between characters.

Syntax :
void pdf_set_word_spacing (int pdf_document, double space)

Deprecated , use pdf_set_value ().

pdf_skew
Rotation of the coordinate system.

Syntax:
void pdf_skew (int pdf_document, double alpha, double beta)

The angle of rotation in degrees is specified relative to the alpha (x) and beta (y) axes. Angles cannot be 90 or 270 degrees.

pdf_continue_text
Displays text on the next line.

Syntax:
void pdf_continue_text (int pdf_document, string text)

The distance between lines can be set with the pdf_set_leading () function.

pdf_stringwidth
Calculates the width of the text.

Syntax:
void pdf_stringwidth (int pdf_document, string text)

The current font is used to calculate the string length. The font must first be installed using pdf_set_font ().
pdf_save
Save current settings.

Syntax:
void pdf_save (int pdf_document)

Acts like the postscript gsave command. Useful when you need to scale or expand an object without affecting other objects. pdf_save () requires that pdf_restore () is then called.

pdf_restore
Restore previously saved settings.

Syntax:
void pdf_restore (int pdf_document)

Restores the settings saved by pdf_save (). Acts like the postscript grestore command.

```
<? php
pdf _ save ($ pdf );
// all sorts of rotations and transformations ...
pdf_restore ($ pdf);
?>
```

Setting the scale and coordinate system

pdf_translate
Sets the origin of the coordinate system.

Syntax:
void pdf_translate (int pdf_document, double x, double y)

Coordinates are relative to the current origin. Then, before you start drawing objects, you need to set the current point.

```
<? php
pdf _ moveto ($ pdf , 0, 0);
pdf_lineto ($ pdf, 100, 100);
pdf_stroke ($ pdf);
psd_translate ($ pdf, 100, 100);
pdf_moveto ($ pdf, 0, 0);
pdf_lineto ($ pdf, 100, 100);
pdf_stroke ($ pdf);
?>
```

pdf_scale
Setting scaling . **Syntax :** void pdf_scale (int pdf_document, double x_scale, double y_scale)

```
<? php
pdf_scale ($ pdf, 72.0, 72.0);
pdf_lineto ($ pdf, 1, 1); // per inch
pdf_stroke ($ pdf);
?>
```

pdf_rotate
Sets the rotation angle in degrees.

Syntax :
void pdf_rotate (int pdf_document, double angle)

pdf_setflat
Set **flatness** .

Syntax:
void pdf_setflat (int pdf_document, double value)

Possible parameter values - from 0 to 100.

pdf_setlinejoin
Sets the linejoin parameter.

Syntax:
void pdf_setlinejoin (int pdf_document, double value)

Possible parameter values are from 0 to 2.

pdf_setlinecap
Sets the linecap parameter.

Syntax:
void pdf_setlinecap (int pdf_document, double value)

Possible parameter values are from 0 to 2.

pdf_setmiterlimit
Setting the miter limit parameter . **Syntax :** void pdf_miterlimit (int pdf_document, double value) Possible parameter values - 1 or more .

pdf_setlinewidth
Sets the width of the lines.

Syntax :
void pdf_setlinewidth (int pdf_document, double width)

pdf_setdash
Sets the current point.

Syntax :
void pdf_setdash (int pdf_document, double white, double black)

pdf_moveto
Sets the current point.

Syntax :
void pdf_moveto (int pdf_document, double x, double y)

Draw and fill shapes

pdf_curveto
Draw the curve . **Syntax :** void pdf_curveto (int pdf_document, double x1, double y1, double x2, double y2, double x3, double y3) Draws a Bézier curve from the current point to (x3, y3), using points (x1, y1) and (x2, y2) as orienting.

pdf_lineto Draw a
segment.

Syntax:
void pdf_lineto (int pdf_document, double x, double y)

Draws a line from the current point to the specified (x, y).

pdf_circle
Draw a circle . **Syntax :** void pdf_circle (int pdf_document, double x, double y, double radius)

pdf_arc Draw an
arc.

Syntax:
void pdf _ arc (int pdf _ document , double x , double y , double radius , double start , double end) The *start* and *end* angles are specified in *start* and *end* .

pdf_rect Draws a
rectangle.

Syntax:
void pdf_rect (int pdf_document, double x, double y, double width, double height) The lower left corner is set to (*x* , *y*); height and width - *height* and *width* .

pdf_closepath
Completion of the current path.

Syntax:
void pdf_closepath (int pdf_document)

Draws a line from the current point to the point where the first line began. Many functions like pdf_moveto (), pdf_circle (), pdf_rect () start a new path.

pdf_stroke Path shading
.

Syntax:
void pdf_stroke (int pdf_document) The

current path is the collection of all lines. Without this function, lines will not be drawn.

pdf_closepath_stroke Draw
and close path.

Syntax:
void pdf_closepath_stroke (int pdf_document)

This is a combination of pdf_closepath () and pdf_stroke ().

pdf_fill
Fill the path with color.

Syntax :
void pdf_fill (int pdf_document)

pdf_fill_stroke
Fill the path with color and close it.

Syntax :
void pdf_fill_stroke (int pdf_document)

pdf_closepath_fill_stroke Draw
, fill and close path.

Syntax :
void pdf _ closepath _ ! Just fill _ stroke of (int pdf _ document)

pdf_endpath
Ending a path without closing it.

Syntax :
void pdf_endpath (int pdf_document)

pdf_clip
Attaches all lines to the current path.

Syntax :
void pdf_clip (int pdf_document)

pdf_setgray_fill
Sets the gray fill.

Syntax :
void pdf_setgray_fill (int pdf_document, double gray_value)

pdf_setgray_stroke
Sets the shading to gray.

Syntax :
void pdf_setgray_stroke (int pdf_document, double gray_value)

pdf_setgray
Sets the fill and hatch to gray.

Syntax :
void pdf_setgray (int pdf_document, double gray_value)

pdf_setrgbcolor_fill
Sets the fill with RGB color.

Syntax :
void pdf_setrgbcolor_fill (int pdf_document, double red_value, double green_value, double blue_value)

pdf_setrgbcolor_stroke
Sets the RGB shading.

Syntax :
void pdf_setrgbcolor_stroke (int pdf_document, double red_value, double green_value, double blue_value)

pdf_setrgbcolor
Sets RGB fill and hatch.

Syntax :
void pdf_setrgbcolor (int pdf_document, double red_value, double green_value, double blue_value)

pdf_add_outline
Add a bookmark for the current page.

Syntax:
void pdf_add_outline (int pdf_document, string text [, int parent [, int open]])

The name of the bookmark is specified by the *text* argument . It becomes a child of the *parent* object and is open by default (if *open is* not 0). The bookmark identifier is returned and can be used as a parent for other

bookmarks.

pdf_set_transition
Sets the transition mode between pages.

Syntax :
void pdf_set_transition (int pdf_document, int transition)

Use the pdf_set_parameter () function with the "transition" parameter .

pdf_set_duration
Sets the spacing between pages.

Syntax :
void pdf_set_duration (int pdf_document, double duration)

Placing Pictures

pdf_open_gif
Opens GIF image.

Syntax :
void pdf_open_gif (int pdf_document, string filename)

Use pdf_open_image_file () function .

```
<? php
$ im = pdf_open_gif ($ pdf, "test.gif");
pdf_place_image ($ pdf, $ im, 100, 100, 1);
pdf_close_image ($ pdf, $ im);
?>
```

pdf_open_png
Opens a PNG image.

Syntax :
void pdf_open_png (int pdf_document, string filename)

Use pdf_open_image_file () function .

pdf_open_jpeg
Opens a JPEG picture.

Syntax :
void pdf_open_jpeg (int pdf_document, string filename)

Use the pdf_open_image_file () function .

pdf_open_tiff
Opens a TIFF drawing.

Syntax :
void pdf_open_tiff (int pdf_document, string filename)

Use pdf_open_image_file () function .

pdf_open_image_file
Reading an image from a file.

Syntax:
void pdf_open_tiff (int pdf_document, format: string,: string filename)

This function loads the format picture *format* from the file *filename* and returns its identifier.
Possible formats:
- PNG
- TIFF
- JPEG
- GIF

```
<? php
$ pim = pdf _ open _ image _ file ($ pdf , " png ", " pic . png ");
pdf_place_image ($ pdf, $ pim, 100, 100, 1);
pdf_close_image ($ pdf, $ pim);
?>
```

This function replaces pdf_open_image (), pdf_open_gif (), pdf_open_tiff (), pdf_open_png ().

pdf_open_memory_image
Opens a picture created by PHP graphics functions.

Syntax:
void pdf_open_memory_image (int pdf_document, int image)

The function takes a descriptor of the image generated by PHP and makes it available to the pdf document. The function returns the id of the pdf picture.

```
<? php
$ im = ImageCreate (100, 100);
$ col = ImageColorAllocate ($ im , 80, 45, 190);
ImageFill ($ im, 10, 10, $ col);
$ pim = pdf_open_memory_image ($ pdf, $ im);
ImageDestroy ($ im);
pdf_place_image ($ pdf, $ pim, 100, 100, 1);
pdf_close_image ($ pdf, $ pim);
?>
```

pdf_close_image
Closing the picture.

Syntax:
void pdf_close_image (int pdf_document, int image)

Closes a picture opened by pdf_open_ () functions.

pdf_get_image_height
Sets the height of the image in pixels.

Syntax :
void pdf_get_image_height (int pdf_document, int image)

pdf_get_image_width
Sets the width of the image in pixels.

Syntax :
void pdf_get_image_width (int pdf_document, int image)

pdf_place_image
Placement of the picture on the page.

Syntax :
void pdf_place_image (int pdf_document, int image, double x, double y, souble scale) Placement

position is given by (*x* , *y*); scale - *scale* .

pdf_put_image
Save the figure in pdf for future use.

Syntax:
void pdf_put_image (int pdf_document, int image)

The function embeds the picture into the document without displaying it. Then the picture can be placed on the page using the pdf_execute_image () function as many times as necessary. Useful when inserting a picture multiple times (reduces file size).

Since version 2.01 of pdflib the function is useless and only displays a warning.

pdf_execute_image Places the
saved image on the page.

Syntax :
void pdf_execute_image (pdf_document int, int image, double x, double y, double scale)
displays drawing , the embedded function pdf_put_image (). Since version 2.01 of pdflib the function is useless and only displays a warning.

```
<? php
$ im = ImageCreate (100, 100);
$ col1 = ImageColorAllocate ($ im, 80, 45, 190);
ImageFill ($ im, 10, 10, $ col1);
$ pim = pdf_open_memory_image ($ pdf, $ im);
pdf_put_image ($ pdf, $ pim);
pdf_execute_image ($ pdf, $ pim, 100, 100, 1);
// 200%
pdf_execute_image ($ pdf, $ pim, 200, 200, 2);
pdf_close_image ($ pdf, $ pim);
?>
```

Document style

pdf_set_border_style
Sets the border style for notes and hyperlinks.

Syntax:
void pdf_set_border_style (int pdf_document, string style, double width)

The *style* argument can be "solid" or "dashed". The width is specified by the *width* argument .

pdf_set_border_color
Sets the border color for notes and hyperlinks.

Syntax:
void pdf_set_border_color (int pdf_document, double red, double green, double blue)

Three color components can take values from 0.0 to 1.0

pdf_set_border_dash
Sets the border style for links and notes.

Syntax:
void pdf_set_border_dash (int pdf_document, double black, double white)

Sets the length of black and white bars of broken lines.

pdf_add_annotation
Add annotation.

Syntax:
void pdf_add_annotation (int pdf_document, double llx, double lly, double urx, double ury, string title, string content)

Note is assumed in the lower left corner (*llx* , *lly*), upper right corner (*urx* , *ury*).

Part 7. Tips on the topic

Disable caching with PHP

Most scripts generate documents that change each time the program is started. Obviously, if the user's browser starts caching such documents, nothing good will come of it.
It is possible to prevent the browser and proxy servers from caching documents using the PHP language, namely the Header () function .
To do this, use the following commands at the beginning of the script:

```
Header ("Expires: Mon, 26 Jul 1997 05:00:00 GMT"); // Date in the past
Header ("Cache-Control: no-cache, must-revalidate"); // HTTP / 1.1
Header ("Pragma: no-cache"); // HTTP / 1.1
        Header ("Last-Modified:" .gmdate ("D, d MYH: i: s"). "GMT");
```

To completely disable caching, you always have to send 4 specified headers, and none of them can be skipped - otherwise either the browser or the Proxy server will not work.

Creating a poll in PHP

First, we need to decide what we will ask visitors about. For example, whether they liked your site or not.
for our voting we need four files:
The first will contain the voting form (form.html).
The second file will be responsible for processing the results (golos.php).
The third will store the voting data (data.txt).
The fourth will be responsible for displaying graphic information (img.php).

For example, we want to ask visitors the following questions:
Your opinion about the site: just super, normal, so-so, I don't care, bad, I haven't seen worse.
In the form.html file , write :

```
<form action = golos.php method = post>
<table cellspacing cellpadding = 0 border = 0>
<tr> <td align = center colspan = 2> <B> Voting : </B> </td> </tr>
<tr> <td align = center colspan = 2> <B> Your opinion about the site : </B> </td> </tr>
<tr> <td> <input type = radio name = otv value = 1 checked> </td>
<td> Just super! </td> </tr>
<tr> <td> <input type = radio name = otv value = 2> </td>
<td> Normal . </td> </tr>
<tr> <td> <input type = radio name = otv value = 3> </td> <
td> So so. </td> </tr>
<tr> <td> <input type = radio name = otv value = 4> </td> <
td> I don't care. </td> </tr>
<tr> <td> <input type = radio name = otv value = 5> </td> <
td> Bad . </td> </tr>
<tr> <td> <input type = radio name = otv value = 6> </td> <
td> I haven't seen worse! </td> </tr>
<tr> <td colspan = 2 align = center>
<input type = submit name = golos value = " Vote "> </td> </tr>
</ table </form>
```

After clicking the *Vote* button, the *otv* variable will be processed by the script in the golos.php file. Initial data must be written to the data.txt file, which will then be read and processed from there. Create a file called data.txt and in a text editor write the following lines into it:

```
Voting results:
0
0
0
0
```

0
0
The first line will not be taken into account.
In the remaining six lines, you must enter zeros, pressing the *Enter* key after each digit .
In the golos.php file, which is responsible for processing the results, write the following:

```
<html>
<head>
<title> Processing Voting </title>
</head>
<body>
<?
if (@ $ golos) {
// Here we start data processing only if
// the Vote key was pressed
  $ file_name = "data.txt";
  // Variable $ file_name sets the name of the file with the results
  $ file = file ($ file_name);
  // Write the data file to the $ file array
  $ file_len = count ($ file);
// $ file_len - number of lines in the data.txt file
for ($ i = 1, $ n = 0; $ i <$ file_len; $ i ++) {
  $ file [$ i] = trim ($ file [$ i]);
  $ n = $ n + $ file [$ i];
};
// In this loop, we remove the line feed characters and write to
// variable $ n how many people have already voted
echo "<center> <h2> Thank you, your opinion was taken into account! </h2> </center>";
$ file [$ otv] ++;
$ n ++;
// Here we take into account the number of the answer that came to us from the form,
// increasing the corresponding value in the array and the number of voters by 1
$ rez = "Voting results: \ n";
// The variable $ rez will contain the voting data,
// which we then write back to the file
for ($ i = 1; $ i <$ file_len; $ i ++) $ rez. = $ file [$ i]. "\ n";
$ rez = trim ($ rez);
$ file_rec = @ fopen ($ file_name, "w");
// Here we create a new file, which we then write the updated data to
if ($ file_rec) {
  $ counter = fputs ($ file_rec, $ rez);
  // Write updated data to data.txt file
  fclose ($ file_rec);
  }
  else echo "An error occurred while writing the results!";
  for ($ i = 1; $ i <$ file_len; $ i ++) $ pr [$ i] = round (($ file [$ i] / $ n) * 100);
// Write to the $ pr array, what percentage of the total number of voters
// takes each answer
// Next is a piece of HTML code that
// responsible for displaying our data on the screen
?>
<center> <h2> Your opinion about the site : </h2> </center> <BR>
<table border = 1 align = center>
<tr> <td>
<B> Just super! (<? echo $ file [1];?>): </B>
</td> <td>
<img src = "img.php? pr = <? echo $ pr [1];?>" height = 15>
</td> </tr>
<tr> <td>
<B> Normal. (<? echo $ file [2];?>): </B>
</td> <td>
```

```
<img src = "img.php? pr = <? echo $ pr [2];?>" height = 15>
</td> </tr>
<tr> <td>
<B> So so. (<? echo $ file [3];?>): </B>
</td> <td>
<img src = "img.php? pr = <? echo $ pr [3];?>" height = 15>
</td> </tr>
<tr> <td>
<B> I don't care. (<? echo $ file [4];?>): </B>
</td> <td>
<img src = "img.php? pr = <? echo $ pr [4];?>" height = 15>
</td> </tr>
<tr> <td>
<B> Bad . (<? echo $ file [5];?>): </B>
</td> <td>
<img src = "img.php? pr = <? echo $ pr [5];?>" height = 15>
</td> </tr>
<tr> <td>
<B> I have not seen worse! (<? echo $ file [6];?>): </B>
</td> <td>
<img src = "img.php? pr = <? echo $ pr [6];?>" height = 15>
</td> </tr>
</table>
<table border = 1 align = center>
<tr> <td align = center> <B> Total voted : </B> </td> </tr>
<tr> <td align = center> <? echo $ n. " person ";?> </td> </tr>
</table>
<?
};
?>
</body>
</html>
```

By writing the value "img.php? Pr = *percentage* " to the SRC attribute of the <IMG> tag , we thereby pass to the img.php file (which is responsible for displaying graphic information) a value based on which the voting image will be generated on the fly. Below is a listing of the img.php file:

```
<?
  $ otstup = 35;
  // $ otstup - sets the indentation in which
  // write a percentage value into the picture
  $ string = $ pr. "%";
  // $ string - contains the percent value plus the percent sign
  $ im = imageCreate ($ pr * 2 + $ otstup, 15);
  // Create an identifier here using
  // which we will work with the picture
  $ fon = imageColorAllocate ($ im, 220,20,60);
  $ fon1 = imageColorAllocate ($ im, 255,20,147);
  // Set the background color
  $ col_b = imageColorAllocate ($ im, 0,0,0);
  // Set the stroke color
  $ shrift = imageColorAllocate ($ im, 255,255,255);
  // Output color of the percentage value
  imageFill ($ im, 2,2, $ fon);
  // Fill our rectangle with the main background
  $ x1 = 0; $ x2 = $ pr * 2 + $ otstup-1;
  $ y1 = 0; $ y2 = 14;
  // Shaping the catch for the outline
  imageLine ($ im, $ x1, $ y1, $ x2, $ y1, $ col_b);
  imageLine ($ im, $ x2, $ y1, $ x2, $ y2, $ col_b);
  imageLine ($ im, $ x2, $ y2, $ x1, $ y2, $ col_b);
  imageLine ($ im, $ x1, $ y1, $ x1, $ y2, $ col_b);
```

```
  imageLine ($ im, $ x1 + $ otstup, $ y1, $ x1 + $ otstup, $ y2, $ col_b);
  // Create an outline and dividing strip
  if ($ pr! = 0) imageFill ($ im, $ otstup + 1,2, $ fon1);
  // If the percentage value is not 0, then fill
  // right side with color $ fon1
  imageString ($ im, 3,5,1, $ string, $ shrift);
  // Write a percentage value to the right side of the picture
  header ("Content-type: image / png");
  imagePng ($ im);
  imageDestroy ($ im);
  // Here we output the resulting image in
  // stdout and destroy the identifier
?>
```

As a result, you get something like:

Your opinion about the site:

Just super! (fifteen):	
Normal. (12):	
So-so. (ten):	
I do not care. (ten):	
Bad. (2):	
I have not seen worse! (1):	

Total Voted:
50 people

Sending emails with PHP

General questions, encoding issues, HTML submission

General features

Sooner or later, every site owner is faced with the need to send letters directly from the site through a script, and not through mail programs. These can be letters sent by the guestbook script, informing the site owner that a new message has appeared in his guestbook, or by a forum to notify about a new question.
In all these cases, it is necessary to automatically send letters bypassing a variety of mail programs and utilities.
This can be done using the mail () function, which we will now study.

The syntax for the mail () function is:

```
bool mail (string $ to, string $ subject, string $ msg [, string $ header]);
```

The mail () function sends an email with subject $ subject and content $ msg to $ to.
If you want the letter to go to several addresses, separate them with spaces.
The message itself can be multi-line. For a newline, put a newline "\ n" at the end of each line.

```
mail ("name@mail.ru", "my subject", "stroka1 \ nstroka2 \ nstroka3");
```

Or the same thing can be written like this:

```
mail ("name@mail.ru", "my subject", "stroka1
stroka2
stroka3 ");
```

In the fourth optional parameter, $ header, you can specify the headers of our message.
By headers, I mean information transmitted along with the letter to the mail client, which will contain some technical data, such as: letter encoding, sender's name, sender's return address, etc. This is similar to using the <META> tag in HTML.

To make it clearer what the headers are and where they are in the letter, open the mail client and look at any letter as it came (For Outlook, this can be done by right-clicking on the message-> Properties-> Details-> Original message).

As we can see, the letter is a simple text file consisting of two general sections:
At the top are the letter headers, then two lines later the letter itself.

There are a lot of email headers, but not all of them are used when writing a mail sending script.
Some of the most commonly used are:

From: " *Sender Username* " < *Sender Return Address* >
To: < *Address to which the letter is sent* >
Subject: Email subject

Solving the problem of encodings

It was not for nothing that I gave an example of sending a letter with words of the Latin alphabet above. Any mail client will read them without difficulty. But with the Russian alphabet it is more difficult. There are many Russian encodings. And it will depend on how skillfully you recode the letter whether the recipient will read it, or will not bother with setting the desired encoding and will simply delete it into the trash.
The message encoding is set by the Content-type header:

```
$ header = "From:" \ Evgen "\ <evgen@mail.ru>";
$ header. = "Content-type: text / plain; charset = \" windows-1251 \ "";
$ subject = "Email subject";
$ msg = "Storaka 1 \ nLine 2 \ nLine 3";
mail ("name@mail.ru", $ subject, $ msg, $ header);
```

In the title, we indicated that the letter type will be plain text, and the encoding will be Windows.
Now our letter will come in an encoding understandable for the mail client.
But it should be noted that in some cases only the letter itself will be displayed in the correct encoding. The headline will remain unreadable. This is due to where the Content-type header is located relative to the Subject header, which contains the email subject. The fact is that there are mailers that understand the Content-type header, but do not understand the Russian text in the Subject field, if this field comes before the Content-type. At the same time, other mailers oblige us to specify the Content-type as the last header in the list. To get around these obstacles, you can place the Content-type field immediately at the beginning and end of the header list:

```
$ subject = " Subject letters ";
$ header = "Content-type: text / plain; charset = \" windows-1251 \ "";
$ header. = "From:" \ Evgen "\ <evgen@mail.ru>";
$ header. = "Subject: $ subject";
$ header. = "Content-type: text / plain; charset = \" windows-1251 \ "";
$ msg = " Storaka 1 \ n Line 2 \ n Line 3";
mail ("name@mail.ru", $ subject, $ msg, $ header);
```

Now this letter can be read by any mail program!

Sending an email in HTML form

To send an email in HTML, it is enough to specify the document type in the Content-type header not text / plain (plain text), but text / html (html-text). And write the letter itself in html-form:

```
$ subject = "Email subject";
$ header = "Content-type: text / html; charset = \" windows-1251 \ "";
$ header. = "From:" \ Evgen "\ <evgen@mail.ru>";
$ header. = "Subject: $ subject";
$ header. = "Content-type: text / html; charset = \" windows-1251 \ "";
$ msg = "<body>
<li> Storaka 1
<li> Storaka 2
<li> Storaka 3
</body> ";
mail ("name@mail.ru", $ subject, $ msg, $ header);
```

Attaching a file

The letter containing the attached file is somewhat different from the simple one. It adds some headers and changes the structure a bit, although the letter itself is undoubtedly a text file. But let's take everything in order.

One of the features is the presence of the **Mime-Version** header .

This header indicates the standard to which the message body conforms.
MIME conformant messages must contain such a header field with the following text:

```
MIME-Version: 1.0
```

If we want to send an email with attached files, then we need to use the **Content-type: multipart / mixed** header , which means that the email consists of several parts, each of which contains its own Content-type header.
To indicate the boundaries of these parts, you must use the boundary parameter, which is also called a boundary marker.
Any string can be used as the value for this parameter. But we must take into account that it must be unique and did not appear in the body of the letter. Otherwise, the letter may be broken into parts incorrectly.

```
From: "Uspenskii Evgeny" <evgeny@spravkaweb.ru>
To: user@domain.ru
Subject: Hello
Mime-Version: 1.0
Content-Type: multipart / mixed; boundary = "spravkaweb-12345"
```

When dividing a letter into parts, two hyphens should be placed before the marker.
And the last marker, which marks the end of the letter, must contain two hyphens at the end.
Each part needs to set its own titles.
After the headings, you must put two line feed characters.

```
From: "Uspenskii Evgeny" <evgeny@spravkaweb.ru>
To: user@domain.ru
Subject: Hello
Mime-Version: 1.0
Content-Type: multipart / mixed; boundary = "spravkaweb-12345"
--spravkaweb-1234
Content-type: text / plain; charset = "windows-1251"
Content-Transfer-Encoding: quoted-printable

Hi!
Here's that my file!

--spravkaweb-1234
Content-Type: application / x-rar-compressed; name = "file.rar"
Content-Transfer-Encoding: base64
Content-Disposition: attachment

UmFyIRoHAM + QcwAADQAAAAAAAABvYXQg .....
spravkaweb-1234--
```

If at us is a part of a text , it is necessary header Content-Transfer-Encoding assign value quoted-printable, or 7bit, or 8bit. For the file part, this header must be base64. The Content-Disposition header, present in the second part, indicates how the mailer should display this part of the letter. It can take the value attachment (this section is not part of the letter, but is simply attached to it as a file) and inline (an inclusion that is used directly in the letter, for example, a picture inserted into HTML). In the first part heading

```
Content-type: text / plain; charset = "windows-1251"
```

indicated that it is plain text encoded by Windows.
In the second as part of the title

```
Content-Type: application / x-rar-compressed; name = "file.rar"
```

indicated that the file type is rar-archive, and the file name is file.rar.
If we send a gif image, its Content-type will look like:

```
Content-Type: image / gif; name = "file.gif"
```

If it is not known in advance what type of file we are sending, or the file format is not standard, set the Content-type header to application / octet-stream.

```
Content-Type: application / octet-stream; name = "file.dat"
```

Attached files must be placed in a letter in base64 format. You can convert a file to this format using the base64_encode () function:

```
// Open the file for reading in binary format
```

```
$ file = fopen ("file.zip", "rb");
// Read it into string $ str_file
$ str_file = fread ($ file, filesize ("file.zip"));
// Convert this string to base64 format
$ str_file = base64_encode ($ str_file);
```

Now the variable *$ str_file* , which contains the file, can be inserted into the letter.

For the final consolidation of the material, we will write a function that sends a letter in HTML format to the specified addressee with an attached file:

```
/ *
$ to - email recipient address
$ from_mail - mail sender address
$ from_name - name of the message sender
$ subject - email subject
$ message - the message itself in HTML format
$ file_name - path to the file to be attached to the letter
(this could be the name of the file selected in the <input type = file name = file_name> field)
* /
function sendMail ($ to, $ from_mail, $ from_name, $ subject, $ message, $ file_name) {
$ bound = "spravkaweb-1234";
$ header = "From: \" $ from_name \ "<$ from_mail> \ n";
$ header. = "To: $ to \ n";
$ header. = "Subject: $ subject \ n";
$ header. = "Mime-Version: 1.0 \ n";
$ header. = "Content-Type: multipart / mixed; boundary = \" $ bound \ "";
$ body = "\ n \ n - $ bound \ n";
$ body. = "Content-type: text / html; charset = \" windows-1251 \ "\ n";
$ body. = "Content-Transfer-Encoding: quoted-printable \ n \ n";
$ body. = "$ message";
$ file = fopen ($ file_name, "rb");
$ body. = "\ n \ n - $ bound \ n";
$ body. = "Content-Type: application / octet-stream;";
$ body. = "name = \" ". basename ($ file_name)." \ "\ n";
$ body. = "Content-Transfer-Encoding: base64 \ n";
$ body. = "Content-Disposition: attachment \ n \ n";
$ body. = base64_encode (fread ($ file, filesize ($ file_name))). "\ n";
$ body. = "$ bound - \ n \ n";
if (mail ($ to, $ subject, $ body, $ header)) {
  echo "<center> Email was sent successfully! </center>";
} else {
  echo "<center> Message not sent! </center>";
};
```

How to insert a picture into an email

Let's say our task is to insert our banner (button) into a letter sent from the site to the user.
This can be done in two ways:

First - in the HTML-code of the letter, we indicate the address of the picture as on a simple page <IMG src = "http://spravkaweb.ru/img/88x31.gif">
In this case, if the user reads your letter online , the picture will be successfully downloaded from the specified server and displayed in the letter. But if the user is not online, the picture will not be able to load.

The second way is to place the picture in the letter itself as an attached file (read more about attaching files to the letter here), assign a unique identifier to this file, and then refer to this identifier in the body of the letter when specifying the address of the picture.

Thus, not only images can be inserted into a letter, but also flash videos, music, ActiveX controls.

Of course, this will increase the size of the letter, but we will be sure that the user will see the inserted image for sure (unless, of course, the display of images is disabled in his email program).

To assign an identifier to a picture, you need to place the following title in the section of the letter where it is located:

Content-ID: < *ID* >

where the *identifier* is a string that will be unique for this message (for example, the boundary parameter of the **Content-Type** header).
Now, in the letter itself, you can substitute its identifier in the address of the picture.

<IMG src = "cid: *id* ">

The mail program will analyze it, extract a picture from the corresponding section and display it.
This is what a letter might look like:

```
Date: Sat, 13 Mar 2004 09:56:31 -0300
Subject: Send image
From: "Evgen" <admin@spravkaweb.ru>
To: admin@localhost.ru
Subject: Send image
Mime-Version: 1.0
Content-Type: multipart / alternative; boundary = "spravkaweb-1234"

--spravkaweb-1234
Content-type: text / html; charset = "windows-1251"
Content-Transfer-Encoding: 8bit

<h3> Hello </h3>
This is a sample of sending a letter with an attached picture. <BR>
And here is the picture itself: <BR>
<img src = "cid: spravkaweb_img_1">

--spravkaweb-1234
Content-Type: image / jpeg; name = "5.jpg"
Content-Transfer-Encoding: base64
Content-ID: <spravkaweb_img_1>

/ 9j / 4AAQSkZJRgABAQAAAQABAAD / 2wBD ....
--spravkaweb-1234--
```

How , and in the case with the attachable files , if we advance not known , a type of image to be fixed , header **Content-Type** , you can assign a value of application / octet-stream. To fix this topic, we will write a program that sends an email with a picture:

```
<? php
/ *
Let's set the path to the inserted image in the $ file_name variable.
In our case, it is located in the same directory as
file sending the letter. But instead, you can substitute here
file , the resulting scenario of the <INPUT type = file name = file_name >.
* /
$ file_name = "5.jpg";
$ subj = "Submit Image";
$ bound = "spravkaweb-1234";
$ headers = "From: \" Evgen \ "<admin@spravkaweb.ru> \ n";
$ headers. = "To: admin@localhost.ru \ n";
$ headers. = "Subject: $ subj \ n";
$ headers. = "Mime-Version: 1.0 \ n";
$ headers. = "Content-Type: multipart / alternative; boundary = \" $ bound \ "\ n";
$ body = "- $ bound \ n";
```

```
$ body. = "Content-type: text / html; charset = \" windows-1251 \ "\ n";
$ body. = "Content-Transfer-Encoding: 8bit \ n \ n";
$ body. = "<h3> Hello </h3>
This is a sample of sending a letter with an attached picture. <BR>
And here is the picture itself: <BR>
<img src = \ "cid: spravkaweb_img_1 \"> ";
$ body. = "\ n \ n - $ bound \ n";
$ body. = "Content-Type: image / jpeg; name = \" ". basename ($ file_name)." \ "\ n";
$ body. = "Content-Transfer-Encoding: base64 \ n";
$ body. = "Content-ID: <spravkaweb_img_1> \ n \ n";
$ f = fopen ($ file_name, "rb");
$ body. = base64_encode (fread ($ f, filesize ($ file_name))). "\ n";
$ body. = "- $ bound - \ n \ n";
mail ("admin@localhost.ru", $ subj, $ body, $ headers);
?>
```

I want to note that in a similar way, you can insert into the letter not only pictures, but, for example, flash videos, sound, and other elements that should be loaded to the page from files.

PHP to Excel: Working with COM Objects

Introduction

This article series focuses on creating Excel documents with PHP. This feature may be needed, for example, if you need to provide the user with downloadable data in the form of Excel sheets. These can be prices for products that are automatically generated from a database on the server, or some documents that also need to be presented in the form of Excel documents.

Here we consider the possibility of working with Excel documents through a COM object. Unfortunately, working with COM objects in PHP is only possible on Windows platforms. Therefore, if you use Unix hosting, then you will not be able to create and edit Excel documents in the following way.

Working with COM objects is done using syntax

```
$ com_object = new COM ( $ object );
```

Where

- $ com_object - new COM object ;
- *$ object* - id-class of the required object.

To create Excel documents, the $ object variable must be set to "Excel.Application" or "Excel.sheet".

```
$ xsl = new COM ("Excel.Application");
```

After creating a new COM object, you can access its properties and methods:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ rangeValue = $ xls-> Range ("A1");
$ rangeValue-> Value = "In the selected block, the text will be bold, underlined, italic";
$ rangeValue = $ xls-> Range ("A2");
$ rangeValue-> Value = " The font will have a height of 12";
$ rangeValue = $ xls-> Range ("A3");
$ rangeValue-> Value = " Font name - Times New Roman";

$ range = $ xls-> Range ("A1: J10"); // Define the area of  the cells
$ range-> Select (); // Select it
$ fontRange = $ xls-> Selection (); // Set the selection to a variable

// Next, set the parameters for formatting the text in the selected area
$ fontRange-> Font-> Bold = true; // Bold
$ fontRange-> Font-> Italic = true; // Italic
$ fontRange-> Font-> Underline = true; // Underlined
```

```
$ fontRange-> Font-> Name = "Times New Roman"; // Font name
$ fontRange-> Font-> Size = 12; // Font size

?>
```

Later in this section I will give examples of working with the main most used properties and methods:

- Opening, recording, closing a document
- Cell Format: Alignment
- Cell format: Font
- Working with strings
- Working with columns
- Add / Delete / Rename Sheets
- Drawing tables
- Copy / paste cells

Opening, recording, closing a document

General features:

In Excel, using PHP, you can perform the following actions with documents:

- create a new document;
- open a previously created document;
- save the open document;
- close the document.

Creating a new document:

Creating a new document in Excel takes three steps:

1. - We create a "link" between PHP and Excel (a descriptor is created, as when working with files);
2. - We indicate whether the program will be visually open or not;
3. - We indicate to the program through the descriptor that you need to open a new document.

To create a descriptor ("connection"), you need to use a call to Excel through a COM object:

```
$ xls = new COM ("Excel.Application");
```

Now, through the *$ xls* descriptor, we can access all the properties and methods of Excel.
Whether Excel is visible or not is specified in the **Visible ()** property of the **Application ()** object .
If we assign the value 1 to this property, then the program will be displayed, if 0, then no:

```
$ xls-> Application-> Visible = 1;
```

Finally, you can add a new document using the **Add ()** method of the **Workbooks ()** object :

```
$ xls-> Workbooks-> Add ();
```

Those. to just run it with PHP Excel, you need to run the following code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document
?>
```

The first two lines of this example should always be used when working with Excel via PHP.

Opening a previously created document:

Opening a document can be done using the **Open ()** method of the **Workbooks ()** object . In the parameter passed to the Open () method, you need to specify the name of the file to open:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Open ("C: \ test.xls"); // Open a previously saved document
?>
```

If you specify not the full but a relative path, then the search for the file to be opened will be performed not on the server but on the user's computer (by default, this is the "My Documents" folder).

Saving an open document:

Saving the open document produced by the method **SaveAs ()** object **the Workbooks ()** :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add ();
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "Trial Write"; // Insert value

// Save the document
$ xls-> Workbooks [1] -> SaveAs ("test.xls");

$ xls-> Quit (); // Close the application
$ xls-> Release (); // Free objects
$ xls = Null;
$ range = Null;
?>
```

Closing a document:

The document is **closed** by the **Quit ()** method .

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add ();
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "Write something down"; // Insert value

// Save the document
$ xls-> Workbooks [1] -> SaveAs ("test.xls");

$ xls-> Quit (); // Close the application
$ xls-> Release (); // Free objects
$ xls = Null;
$ range = Null;
?>
```

Cell Format: Alignment

In Excel, using PHP, you can align data in cells both vertically and horizontally.

General features:

The following actions are available:

- alignment by value;
- left alignment;
- center alignment;
- right alignment;
- fill alignment;
- justified alignment;
- distribution in the center of the selection;
- horizontal distribution;
- top alignment;
- vertical centering;
- bottom alignment;
- height alignment;
- vertical distribution;
- setting cell indentation.

Horizontal alignment:

The **HorizontalAlignment ()** property is responsible for horizontal alignment .
You can align a value in a single cell or in a range of cells. To do this, select a cell (or a range of cells) and set the

HorizontalAlignment () property to one of the 8 predefined values:
• *HorizontalAlignment = 1*
- alignment by value (used by default);
• *HorizontalAlignment = 2*
- alignment to the left;
• *HorizontalAlignment = 3*
- center alignment;
• *HorizontalAlignment = 4*
- right alignment;
• *HorizontalAlignment = 5*
- alignment with filling;
• *HorizontalAlignment = 6*
- justified alignment;
• *HorizontalAlignment = 7*
- center alignment of the selection;
• *HorizontalAlignment = 8*
- horizontal distribution.
In the example below, the top line of the "1: 1" table will align all values to the center.

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Center align
$ rangeAlignment = $ xls-> Range ("1: 1");
$ rangeAlignmeny-> HorizontalAlignment = 3;
$ rangeAlignment-> Value = "Hello";
?>
```

Vertical alignment:

The **VerticalAlignment ()** property is responsible for vertical alignment .
As well as horizontally, you can align a value in one cell or in a range of cells. To do this, select a cell (or a range of cells), and assign one of 5 predefined values to the VerticalAlignment () property:
• *VerticalAlignment = 1*
- top alignment;
• *VerticalAlignment = 2*
- center alignment;
• *VerticalAlignment = 3*
- bottom alignment (default);
• *VerticalAlignment = 4*
- height alignment;
• *VerticalAlignment = 5*
- vertical distribution.
In the example below, the top line of the "1: 1" table will align all values to the top.

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Center align
$ rangeAlignment = $ xls-> Range ("1: 1");
// Set the alignment for the cells: top edge
$ rangeAlignment-> VerticalAlignment = 1;
// Set the font size: 8
$ rangeAlignment-> Font-> Size = 8;
// Set the line height: 25
$ rangeAlignment-> RowHeight = 25;
// Display the inscription : Hello
$ rangeAlignment-> Value = "Hello";
?>
```

Setting the indentation:

The cell indentation is set using the **IndentLevel ()** property .
This property equates to a value equal to the number of characters by which the data should be shifted to the left.

In the following example, we will write the text in cell *A1* with an indent of 5 characters, and in cell *A2* - without an indent:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the indent
$ rangeAlignment = $ xls-> Range ("A1");
$ rangeAlignment-> IndentLevel = 5;
$ rangeAlignment-> Value = "The indent is 5 characters ";

// No indentation
$ rangeAlignment = $ xls-> Range ("A2");
$ rangeAlignment-> Value = "No Indent";
?>
```

Cell format: Font

General features:

In Excel, using PHP with text written in cells, you can do the following:

- make it bold;
- highlight in italics;
- make it underlined;
- install the font;
- set the font size;
- set the color of the text;

Text formatting:

The technique for accessing text properties is as follows:

- first, in a previously created document, set the range of cells to which text formatting will be applied;
- select it;
- assign the selected range of values to the variable;
- using the **Font ()** object, we get access to the formatting of the selection.

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ rangeValue = $ xls-> Range ("A1");
$ rangeValue-> Value = "In the selected block, the text will be bold, underlined, italic";
$ rangeValue = $ xls-> Range ("A2");
$ rangeValue-> Value = " The font will have a height of 12";
$ rangeValue = $ xls-> Range ("A3");
$ rangeValue-> Value = " Font name - Times New Roman";

$ range = $ xls-> Range ("A1: J10"); // Define the area of the cells
$ range-> Select (); // Select it
$ fontRange = $ xls-> Selection (); // Set the selection to a variable

// Next, set the parameters for formatting the text in the selected area
$ fontRange-> Font-> Bold = true; // Bold
$ fontRange-> Font-> Italic = true; // Italic
$ fontRange-> Font-> Underline = true; // Underlined
$ fontRange-> Font-> Name = "Times New Roman"; // Font name
```

```
$ fontRange-> Font-> Size = 12; // Font size

?>
```

Setting text color:

The color of the text is set using the **ColorIndex ()** property of the same **Font ()** object .

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ range = $ xls-> Range ("A1");
$ range-> Value = "Text will be written in red";
$ range-> Font-> ColorIndex = 3;
?>
```

There is one peculiarity: the color of the text is not specified in any format (RGB, CMYK, etc.), but the number under which it is located in the Excel palette.

There are 55 colors in total in the palette and one value is reserved for auto color. In total, 56 color values can be set using the ColorIndex () property.

The following example will print all possible values for the ColorIndex () property in an Excel document:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ min_color_index = 0; // Initial color index
$ max_color_index = 55; // End color index
$ start_position = 2; // Position number from which the display will start
                      // color indices
// Display the inscription "Color number"
$ range = $ xls-> Range ("A1: B1");
$ range-> Font-> Bold = true;
$ range = $ xls-> Range ("A1");
$ range-> Value = "Color numbers";

// Print out the color index values and color these values
// appropriate color
for ($ i = $ min_color_index; $ i <= $ max_color_index; $ i ++) {
  $ range = $ xls-> Range ("A". ($ i + $ start_position));
  $ range-> Value = "ColorIndex =". $ i;
  $ range-> Font-> ColorIndex = $ i;
};
?>
```

Working with strings

General features:

In Excel, using PHP, you can perform the following actions with strings:

- add a line;
- delete a line;
- set the height of one line or group of lines;
- auto-fit the height of one line or a group of lines;
- hide a line or group of lines;
- display a previously hidden row or group of rows.

Adding a line:

You can add a row using the **Insert ()** method of the **EntireRow ()** object :

```
<? php
```

```
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Insert the values  in the 1st, 2nd and 3rd cells
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "1st line"; // Insert value
$ range = $ xls-> Range ("A2"); // Selected cell A2
$ range-> Value = "2nd line; // Insert value
$ range = $ xls-> Range ("A3"); // Selected cell A3
$ range-> Value = "3rd line"; // Insert value

// Insert the line
$ range = $ xls-> Range ("2: 2"); // Determine the location
$ range-> EntireRow-> Insert (); // Insert the line
?>
```

Deleting a line:

A row is **deleted** using the **Delete ()** method of the **EntireRow ()** object :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Insert the values  in the 1st, 2nd and 3rd cells
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "1st line"; // Insert value
$ range = $ xls-> Range ("A2"); // Selected cell A2
$ range-> Value = "2nd line"; // Insert value
$ range = $ xls-> Range ("A3"); // Selected cell A3
$ range-> Value = "3rd line"; // Insert value

// Delete the line
$ range = $ xls-> Range ("2: 2"); // Define a string
$ range-> EntireRow-> Delete (); // Delete it
?>
```

Setting the line height:

You can set the row height using the **RowHeight** property . The height is specified in millimeters.
The following example sets the string "2: 2" to a height of 25 mm:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Change the line height
$ range = $ xls-> Range ("2: 2"); // Select the 2nd line
$ range-> Select (); // Select it
$ rowRange = $ xls-> Selection; // Define $ rowRange as the selection
$ rowRange-> RowHeight = 25; // Set the line height
?>
```

By analogy, you can set the height for several lines:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Change the line height
$ range = $ xls-> Range ("2: 7"); // Select lines 2 to 7
$ range-> Select (); // Select them
```

```
$ rowRange = $ xls-> Selection; // Define $ rowRange as the selection
$ rowRange-> RowHeight = 25; // Set the height of the lines
?>
```

Auto-fit line heights:

Auto-fit for line heights is used for better readability of the displayed data. Those. if the font size of the text placed in the lines is much less than the line height, or much more than the height, then such text is not very pleasant to read.

The AutoFit **()** method of the **Rows** object is used for **auto- fitting** :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ range = $ xls-> Range ("B1"); // Set the 1st cell
$ range-> Font-> Size = 20; // Set the font size
// Output the value in the 1st selected cell
$ range-> Value = "Web-languages  reference: www.spravkaweb.ru";

$ range = $ xls-> Range ("B2"); // Set the 2nd cell
$ range-> Font-> Size = 20; // Set the font size
// Output the value to the 2nd selected cell
$ range-> Value = "Web-languages  reference: www.spravkaweb.ru";

$ range = $ xls-> Range ("1: 2"); // Set 2 lines to work
$ range-> Select (); // Highlight these lines
$ rangeRows = $ xls-> Selection (); // Set $  rangeRows as selection
$ rangeRows-> RowHeight = 15; // Set the line height = 15mm

$ rowRange = $ xls-> Range ("2: 2"); // Highlight the 2nd line

$ rowRange-> Rows-> AutoFit (); // Do auto-fit height
                                // for the third line
?>
```

Those. we got the following:

On the first and second lines, we wrote down the text *Web-languages reference: www.spravkaweb.ru* . The font for the text was set to 20. Then the line height was set to 15, and the second line was auto-sized. As a result, the first line is displayed incorrectly (the upper part of the letters is not visible), and the second is normal.

Hiding / showing lines:

The **Hidden ()** property of the **EntireRow ()** object is responsible for the visual display of rows .

If you set this property to True, then the selected rows will be hidden, if False, then shown.

The following example will hide lines 5 through 10:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ range = $ xls-> Range ("5:10"); // Select lines 5 through 10

$ range-> EntireRow-> Hidden = True; // Hide selected rows
?>
```

Working with columns

General features:

In Excel, using PHP, you can perform the following actions with columns:

- add a column;
- delete a column;

- set the width of one or more columns;
- auto-fit one or more columns;
- hide a column or group of columns;
- display a previously hidden column or group of columns;
- set the width for all columns.

Adding a column:

You can add a column using the **Insert ()** method of the **EntireColumn ()** object :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Insert into the first and second top cell values
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "1st column"; // Insert value
$ range = $ xls-> Range ("B1"); // Selected cell B1
$ range-> Value = "2nd column"; // Insert value

// Add a column
$ range = $ xls-> Range ("B: B"); // Define the horse
$ range-> EntireColumn-> Insert (); // Insert a new column in its place

// Write new value
$ range = $ xls-> Range ("B1"); // Select the second top cell
$ range-> Value = "Insert"; // Write a value to it
?>
```

Removing a column:

A column is **deleted** using the **Delete ()** method of the **EntireColumn ()** object :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Insert the values  in the 1st, 2nd and 3rd cells
$ range = $ xls-> Range ("A1"); // Selected cell A1
$ range-> Value = "1st column"; // Insert value
$ range = $ xls-> Range ("B1"); // Selected cell B1
$ range-> Value = "2nd column"; // Insert value
$ range = $ xls-> Range ("C1"); // Selected cell C1
$ range-> Value = "3rd column"; // Insert value


// Delete the column
$ range = $ xls-> Range ("B: B"); // Define the horse
$ range-> EntireColumn-> Delete (); // Delete it
?>
```

Column width setting:

The principle of setting the width of a column or group of columns is generally similar to setting the height of rows , except that the width is **controlled by the ColumnWidth ()** property and not by **RowHeight ()** . The height is specified in millimeters.

The following example sets column "A: A" to a width of 40 mm:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Change the column width
$ range = $ xls-> Range ("A: A"); // Select the 1st column
```

```
$ range-> Select (); // Select it
$ cellRange = $ xls-> Selection; // Define $ cellRange as the selection
$ cellRange-> ColumnWidth = 40; // Set the column width
?>
```

By analogy, you can set the width for several columns

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Change the width of the columns
$ range = $ xls-> Range ("A: E"); // Select columns A through E
$ range-> Select (); // Select them
$ cellRange = $ xls-> Selection; // Define $ cellRange as the selection
$ cellRange-> ColumnWidth = 40; // Set the width of the columns
?>
```

Auto-fit column widths:

Auto-fitting column widths are used for better readability of the displayed data. Those. if the text length is much larger or much less than the column width, then it is possible to "fit" the column width to this text.
To quickly auto-used method **AutoFit ()** object **the Columns ()** :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ range = $ xls-> Range ("A1"); // Defining the 1- th cell
$ range-> Font-> Size = 20; // Set the font size
// Output the value in the 1st selected cell
$ range-> Value = "Web-languages  reference: www.spravkaweb.ru";

$ range = $ xls-> Range ("B2"); // Set the 2nd cell
$ range-> Font-> Size = 20; // Set the font size
// Output the value to the 2nd selected cell
$ range-> Value = "Web-languages  reference: www.spravkaweb.ru";

$ range = $ xls-> Range ("A: B"); // Set 2 lines to work
$ range-> Select (); // Highlight these lines
$ rangeCells = $ xls-> Selection (); // Set $  rangeCells as selection
$ rangeCells-> ColumnWidth = 5; // Set Column Width = 5mm

$ rangeCells = $ xls-> Range ("B: B"); // Highlight the 2nd column
$ rangeCells-> Columns-> AutoFit (); // Do auto-fit width
                                     // for the second column
?>
```

Those. we got the following:
In cells *A1* and *B2*, we wrote the text *Web-languages reference: www.spravkaweb.ru* . The font for the text was set to 20. Then we set the width of the columns to 5, and auto-fit the width of the second column. As a result, the first column remained 5 mm wide, and the second in width stretched over the entire length of the inscription.

Hiding / showing columns:

The **Hidden ()** property of the **EntireColumn ()** object is responsible for the visual display of columns .
If you set this property to True, then the selected columns will be hidden, if False, then shown.
The following example will hide columns B through D:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ range = $ xls-> Range ("B: D"); // Select columns B through D
```

```
$ range-> EntireColumn-> Hidden = True; // Hide selected columns
?>
```

Setting the width for all columns:

If it becomes necessary to set the width for all columns different from the default, then this can be done using the **StandardWidth ()** property of the **ActiveSheet ()** object :

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the width for all columns
$ xls-> ActiveSheet-> StandardWidth = 5;
?>
```

Add / Delete / Rename Sheets

General features:

In Excel, using PHP, you can perform the following actions with sheets:

- select a sheet;
- add a new sheet;
- move the sheet;
- remove the sheet;
- rename the sheet;

Working with sheets:

Sheets in Excel are accessed through the **Sheets** object .
The **Select ()** method allows you to make a particular sheet of the document active:

```
$ rangeSheet = $ xls-> Sheets (" Sheet 2");
$ rangeSheet-> Select ();
```

where "Sheet2" is the name of the sheet to be active.

You can add a new sheet using the **Add ()** method :

```
$ rangeSheet = $ xls-> Sheets;
$ rangeSheet-> Add ();
```

If this is the first addition of a new sheet, then it is named "Sheet4". Usually a sheet is added before the active sheet. Those. if the Select () sheet selection method was not called before calling the Add () method, then the new sheet is added to the beginning of the sheet list.

The sheet name is contained in the **Name ()** property of the **Sheets ()** object .
In the following example, create a new document and rename the sheets "Sheet1", "Sheet2" and "Sheet3" (which are created automatically) to "Prices", "Contact information", "Order", respectively:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

$ rangeSheet = $ xls-> Sheets (" Sheet 1");
$ rangeSheet-> Name = " Prices ";
$ rangeSheet = $ xls-> Sheets (" Sheet 2");
$ rangeSheet-> Name = " Contact Information ";
$ rangeSheet = $ xls-> Sheets (" Sheet 3");
$ rangeSheet-> Name = " Order ";
?>
```

Drawing tables

General features:

In Excel, using PHP, you can do the following operations with tables:

- set the thickness and style of the table border lines;
- set the thickness and style of the lines of the internal grid of the table;
- set the color of the border and the inner grid of the table;

Working with a table:

All table properties are located in the **Borders ()** object . Or rather, not in an object, but in an array of Borders [] objects. Each element of this array is responsible for a specific part of the table (top border of the table, bottom, inner lines, etc.). And already each element of the array has its own properties that are defined only for this object:

Thus, if we want to draw a plate with a bold outer border in blue and thin inner lines in red, then we must execute the following code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set the properties of the left side of the table
$ rangeBordersLeft = $ range-> Borders ("7");
$ rangeBordersLeft-> LineStyle = 1;
$ rangeBordersLeft-> Weight = 3;
$ rangeBordersLeft-> ColorIndex = 5;
// Set the properties of the table top wall
$ rangeBordersTop = $ range-> Borders ("8");
$ rangeBordersTop-> LineStyle = 1;
$ rangeBordersTop-> Weight = 3;
$ rangeBordersTop-> ColorIndex = 5;
// Set the properties of the bottom wall of the table
$ rangeBordersBottom = $ range-> Borders ("9");
$ rangeBordersBottom-> LineStyle = 1;
$ rangeBordersBottom-> Weight = 3;
$ rangeBordersBottom-> ColorIndex = 5;
// Set the properties of the right side of the table
$ rangeBordersRight = $ range-> Borders ("10");
$ rangeBordersRight-> LineStyle = 1;
$ rangeBordersRight-> Weight = 3;
$ rangeBordersRight-> ColorIndex = 5;
// Set the properties of the inner vertical lines
$ rangeBordersVertical = $ range-> Borders ("11");
$ rangeBordersVertical-> LineStyle = 1;
$ rangeBordersVertical-> Weight = 2;
$ rangeBordersVertical-> ColorIndex = 3;
// Set the properties of the inner horizontal lines
$ rangeBordersHorizontal = $ range-> Borders ("12");
$ rangeBordersHorizontal-> LineStyle = 1;
$ rangeBordersHorizontal-> Weight = 2;
$ rangeBordersHorizontal-> ColorIndex = 3;
?>
```

T . e . first the left side of the table is taken

```
$ rangeBordersLeft = $ range-> Borders ("7");
```

For her, they are asked : line type (solid),

```
$ rangeBordersLeft-> LineStyle = 1;
```

line thickness (bold),

```
$ rangeBordersLeft-> Weight = 3;
```

line color (blue)

```
$ rangeBordersLeft-> ColorIndex = 5;
```

We do the same for the right, top, bottom walls.
For internal vertical lines

```
$ rangeBordersVertical = $ range-> Borders ("11");
```

and internal horizontal lines

```
$ rangeBordersHorizontal = $ range-> Borders ("12");
```

set the line type to solid, line thickness to normal, line color to red:

```
$ rangeBordersVertical-> LineStyle = 1;
$ rangeBordersVertical-> Weight = 2;
$ rangeBordersVertical-> ColorIndex = 3;
```

and

```
$ rangeBordersHorizontal-> LineStyle = 1;
$ rangeBordersHorizontal-> Weight = 2;
$ rangeBordersHorizontal-> ColorIndex = 3;
```

The **LineStyle ()** property **(** line type) can take values from 1 to 13, and the **Weight** property (thickness) can take values from 1 to 4.
How the lines will look with different values of the LineStyle and Weight properties is shown in the following table:

LineStyle	Weight = 1	Weight = 2	Weight = 3	Weight = 4
LineStyle = 1				
LineStyle = 2				
LineStyle = 3				
LineStyle = 4				
LineStyle = 5				
LineStyle = 6				
LineStyle = 7				
LineStyle = 8				
LineStyle = 9				
LineStyle = 10				
LineStyle = 11				
LineStyle = 12				
LineStyle = 13				

An example of forming such a table in Excel using PHP:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the whole range to center align
// font size: 8
// column width: 12
$ range = $ xls-> Range ("A1: E14");
$ range-> HorizontalAlignment = 3;
$ range-> Font-> Size = 8;
$ range-> ColumnWidth = 12;

// Form the " header "
$ range = $ xls-> Range ("A1");
$ range-> Font-> Bold = true;
$ range-> Value = "LineStyle";

$ range = $ xls-> Range ("B1");
$ range-> Font-> Bold = true;
$ range-> Value = "Weight = 1";
```

```
$ range = $ xls-> Range ("C1");
$ range-> Font-> Bold = true;
$ range-> Value = "Weight = 2";

$ range = $ xls-> Range ("D1");
$ range-> Font-> Bold = true;
$ range-> Value = "Weight = 3";

$ range = $ xls-> Range ("E1");
$ range-> Font-> Bold = true;
$ range-> Value = "Weight = 4";

// For each column, display the bottom border of the cell
// with the appropriate values  for the LineStyle and Width properties
for ($ i = 1; $ i <= 13; $ i ++) {
  $ range = $ xls-> Range ("A". ($ i + 1));
  $ range-> Value = "LineStyle = $ i";
  $ range = $ xls-> Range ("B". ($ i + 1));
  $ rangeBordersRight = $ range-> Borders ("9");
  $ rangeBordersRight-> LineStyle = $ i;
  $ rangeBordersRight-> Weight = 1;
};

for ($ i = 1; $ i <= 13; $ i ++) {
  $ range = $ xls-> Range ("C". ($ i + 1));
  $ rangeBordersRight = $ range-> Borders ("9");
  $ rangeBordersRight-> LineStyle = $ i;
  $ rangeBordersRight-> Weight = 2;
};

for ($ i = 1; $ i <= 13; $ i ++) {
  $ range = $ xls-> Range ("D". ($ i + 1));
  $ rangeBordersRight = $ range-> Borders ("9");
  $ rangeBordersRight-> LineStyle = $ i;
  $ rangeBordersRight-> Weight = 3;
};

for ($ i = 1; $ i <= 13; $ i ++) {
  $ range = $ xls-> Range ("E". ($ i + 1));
  $ rangeBordersRight = $ range-> Borders ("9");
  $ rangeBordersRight-> LineStyle = $ i;
  $ rangeBordersRight-> Weight = 4;
};

?>
```

To consolidate the material, we will consider how to implement the formation of table borders in PHP using the example of some standard Excel tools.

How it works:

Removes all borders, frames, etc. in the selected range.

How to implement:

Set the display style for all walls and internal lines: none (LineStyle = Null)

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the entire table
$ rangeDiagonalDown = $ range-> Borders ("5");
$ rangeDiagonalDown-> LineStyle = Null;
$ rangeDiagonalUp = $ range-> Borders ("6");
$ rangeDiagonalUp-> LineStyle = Null;
$ rangeBordersLeft = $ range-> Borders ("7");
$ rangeBordersLeft-> LineStyle = Null;
$ rangeBordersTop = $ range-> Borders ("8");
$ rangeBordersTop-> LineStyle = Null;
$ rangeBordersBottom = $ range-> Borders ("9");
$ rangeBordersBottom-> LineStyle = Null;
$ rangeBordersRight = $ range-> Borders ("10");
$ rangeBordersRight-> LineStyle = Null;
$ rangeBordersVertical = $ range-> Borders ("11");
$ rangeBordersVertical-> LineStyle = Null;
$ rangeBordersHorizontal = $ range-> Borders ("12");
$ rangeBordersHorizontal-> LineStyle = Null;

// ..... some further action .....
?>
```

How it works: Draws the bottom border of the **selection with a** solid line of single thickness.

How to implement:
Set the line style for the bottom border: solid (LineStyle = 1), thickness: single (Weight = 2)

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the bottom wall of the table
$ rangeBordersBottom = $ range-> Borders ("9");
$ rangeBordersBottom-> LineStyle = 1;
$ rangeBordersBottom-> Weight = 2;

// ..... some further action .....
?>
```

How it works: Draws the left border of the **selection with a**

solid line of single thickness.

How to implement:
Set the line style for the left border: solid (LineStyle = 1), thickness: single (Weight = 2)

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the left side of the table
$ rangeBordersLeft = $ range-> Borders ("7");
$ rangeBordersLeft-> LineStyle = 1;
$ rangeBordersLeft-> Weight = 2;

// ..... some further action .....
?>
```

How it works: Draws the right border of the **selection with a** solid line of single thickness.

How to implement:
Set the line style for the right border: solid (LineStyle = 1), thickness: single (Weight = 2)

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the right side of the table
$ rangeBordersRight = $ range-> Borders ("10");
$ rangeBordersRight-> LineStyle = 1;
$ rangeBordersRight-> Weight = 2;

// ..... some further action .....
?>
```

How it works: Draws the bottom border of the **selection with a** solid double line.

How to implement:
Set the line style for the bottom border: double (LineStyle = 9), thickness: Weight = 4

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document
```

```
// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the right side of the table
$ rangeBordersRight = $ range-> Borders ("10");
$ rangeBordersRight-> LineStyle = 9;
$ rangeBordersRight-> Weight = 4;

// ..... some further action .....
?>
```

Principle of action: Draws the bottom border of the **selection with a** solid line of double thickness.

How to implement:

Set the line style for the bottom border: single (LineStyle = 1), thickness: Weight = 3

The code:

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the table area
$ range = $ xls-> Range ("B2: E10");
// Select it
$ range-> Select ();
// Assign the selection to the $ range variable
$ range = $ xls-> Selection ();

// Set properties for the right side of the table
$ rangeBordersRight = $ range-> Borders ("10");
$ rangeBordersRight-> LineStyle = 1;
$ rangeBordersRight-> Weight = 3;

// ..... some further action .....
?>
```

Copying / pasting cells

In Excel, using PHP, you can copy cells and paste previously copied cells.

General features:

- copying cells;
- pasting copied cells;
- moving cells;

Copying / pasting cells:

Copying cells occurs in two stages: first, you need to select the copied area and copy it using the **Copy ()** method , and then you need to select the area into which the copied cells will be pasted and paste them

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the copied cell
```

```
$ range = $ xls-> Range ("A1");
$ range-> Value = "Web Language Reference";
// Copy it
$ range-> Copy ();
// Set the area where the cell will be copied
$ range = $ xls-> Range ("A3: A9");
$ range-> Select ();
$ range = $ xls-> Selection ();
// Insert
$ xls-> ActiveSheet-> Paste ();
?>
```

Here **ActiveSheet** is a link to the selected cells .

Moving cells:

If during copying the copied cells remained in place, then when moving the copied cells are deleted.
Moving is performed by the **Cut ()** method .

```
<? php
$ xls = new COM ("Excel.Application"); // Create a new COM object
$ xls-> Application-> Visible = 1; // Make it display
$ xls-> Workbooks-> Add (); // Add a new document

// Set the copied cell
$ range = $ xls-> Range ("A1");
$ range-> Value = "Web Language Reference";
// Copy it
$ range-> Cut ();
// Set the area where the cell will be copied
$ range = $ xls-> Range ("A3");
$ range-> Select ();
$ range = $ xls-> Selection ();
// Insert
$ xls-> ActiveSheet-> Paste ();
?>
```