Array Manipulation With PHP 4

By Vikram Vaswani

Table of Contents

Knowledge Is Power

As a PHP developer myself, I spend a fair part of my time looking for ways to write cleaner, more efficient code. I'll be the first to admit that I don't succeed too often; however, that doesn't stop me from making the effort. And nine times out of ten, I don't have to look too far − the PHP community is incredibly generous with its advice and assistance, and the language itself offers tools which can slice through complex problems in much the same way as a hot knife slices through butter.

Nowhere is this clearer than in the case of PHP's built−in functions, which offer developers power and flexibility of the sort not seen in most other languages. Sadly, most developers don't use this power − not because they don't want to, but simply because they're not even aware of the full extent of PHP's capabilities. And this lack of knowledge can often spell the difference between a tight, efficient Web application and a slow, inefficient one.

Over the next few pages, I'm going to illustrate my point by taking an in−depth look at PHP's array manipulation capabilities. If you're like most developers, you probably use arrays extensively in your development activities; however, your knowledge of array manipulation techniques is limited to counting the elements of an array or iterating through key−value pairs.

PHP's array manipulation functions allow you to do much, much more...and this article will open your eyes to the possibilities.

In addition to providing a gentle introduction to PHP programming in general (and PHP array manipulation in particular), this article will offer you a broad overview of PHP's array functions, serving as both a handy reference and a tool to help you write more efficient code. Regardless of whether you're new to PHP or if you've been working with the language for a while, you should find something interesting in here.

Let's get started!

Just Friends

We'll begin right at the top, with some very basic definitions and concepts.

Unlike string and numeric variables, which typically hold a single value, an array can best be thought of as a "container" variable, which can contain one or more values. For example,

```
<?
$friends = array("Rachel", "Monica", "Phoebe", "Joey",
"Chandler", "Ross");
?>
```

Here, $friends is an array variable, which contains the values "Rachel", "Monica", "Phoebe", "Joey", "Chandler", and "Ross".

Array variables are particularly useful for grouping related values together − names, dates, phone numbers of ex−girlfriends et al.

The various elements of the array are accessed via an index number, with the first element starting at zero. So, to access the element

```
"Rachel"
```

I would use the notation

```
$friends[0]
```

while

```
"Joey"
```

would be

```
$friends[3]
```

− essentially, the array variable name followed by the index number enclosed within square braces. Geeks refer to this as "zero−based indexing".

PHP also allows you to replace indices with user−defined "keys", in order to create a slightly different type of array. Each key is unique, and corresponds to a single value within the array.

```
<?
$starwars = array("new hope" => "Luke", "teacher" => "Yoda",
"bad guy" =>
"Darth");
?>
```

In this case, $starwars is an array variable containing three key−value pairs. The => symbol is used to indicate the association between a key and its value.

In order to access the value

```
"Luke"
```

I would use the key "new hope"

```
$starwars["new hope"]
```

while the value

```
"Darth"
```

would be accessible via the key "bad guy"

```
$starwars["bad guy"]
```

This type of array is sometimes referred to as a "hash" or "hash table" − if you've ever used Perl, you'll see the similarities to the Perl hash variable.

PHP arrays can store strings, number and other arrays. In fact, the same array can store multiple types of data, a luxury not available in many other programming languages. Here's an example:

```
<?
// create array
$all_mixed_up = array(
"numbers" => array("one", 2, 3, "four"),
```

```
"superheroes" => array("spiderman", "superman", "captain
marvel"),
"icecream" => array("chocolate" => "brown", "vanilla" =>
"white",
"strawberry" => "pink"),
);

// returns null - no element with key 2
echo $all_mixed_up[2];

// returns "Array"
echo $all_mixed_up["numbers"];

// returns "one"
echo $all_mixed_up["numbers"][0];

// returns "captain marvel"
echo $all_mixed_up["superheroes"][2];

// returns "white"
echo $all_mixed_up["icecream"]["vanilla"];
?>
```

Having Your Cake

As you may have already guessed, defining an array variable takes place via the array() function − here's how:

```
<?
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake");
?>
```

The rules for choosing an array variable name are the same as those for any other PHP variable − it must begin with a letter, and can optionally be followed by more letters and numbers.

Alternatively, you can define an array by specifying values for each element in the index notation, like this:

```
<?
$desserts[0] = "chocolate mousse";
$desserts[1] = "tiramisu";
$desserts[2] = "apple pie";
$desserts[3] = "chocolate fudge cake";
?>
```

Incidentally, you can omit the index numbers if you prefer − the following snippet is equivalent to the one above:

```
<?
$desserts[] = "chocolate mousse";
$desserts[] = "tiramisu";
$desserts[] = "apple pie";
$desserts[] = "chocolate fudge cake";
?>
```

If you'd prefer to use keys instead of numeric indices, the following examples should make you happier:

```
<?
$movies = array("romance" => "Moulin Rouge", "epic" =>
"Gladiator",
"action" => "The Terminator");

// this is equivalent
$movies["romance"] = "Moulin Rouge";
```

```
$movies["epic"] = "Gladiator";
$movies["action"] = "The Terminator";
?>
```

You can use the range() function to automatically create an array containing a range of elements:

```
<?
// returns the array ("30", "31", "32", "33", "34", "35",
"36", "37", "38",
"39", "40")
$thirties = range(30, 40);

// returns the array ("i", "j", "k", "l", "m", "n", "o")
$alphabet = range("i", "o");
?>
```

You can add elements to the array in a similar manner − for example, if you wanted to add the element "apricot fritters" to the $desserts array, you would use this:

```
<?
$desserts[4] = "apricot fritters";
?>
```

and the array would now look like this:

```
<?
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake", "apricot fritters");
?>
```

The same goes for non−numeric keys − adding a new key to the $movies array

```
<?
$movies["horror"] = "The Sixth Sense";
?>
```

alters it to read:

```
<?
$movies = array("romance" => "Moulin Rouge", "epic" =>
"Gladiator",
"action" => "The Terminator", "horror" => "The Sixth Sense");
?>
```

In order to modify an element of an array, you would simply assign a new value to the corresponding scalar variable. If you wanted to replace "chocolate fudge cake" with "chocolate chip cookies", you'd simply use

```
<?
$desserts[3] = "chocolate chip cookies";
?>
```

and the array would now read

```
<?
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
chip cookies", "apricot fritters");
?>
```

You can do the same with named keys − the statement

```
<?
$movies["action"] = "Rambo";
?>
```

further alters the $movies array to

```
<?
$movies = array("romance" => "Moulin Rouge", "epic" =>
"Gladiator",
"action" => "Rambo", "horror" => "The Sixth Sense");
?>
```

If it walks like an array, talks like an array and looks like an array, it must be an array...right? Well, you can always verify your suspicions with the is_array() function, which comes in handy if you need to test whether a particular variable is an array or not.

```
<?
// create array
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake", "apricot fritters");

// returns true
echo is_array($desserts);

$desserts = "blueberry ice cream";

// returns false
echo is_array($desserts);
?>
```

When Size Does Matter...

Once an array has been created, it's time to use it. Within the context of an array, the most important (and commonly−used) function is the sizeof() function, which returns the size of (read: number of elements within) the array.

Here's an example:

```
<?
// create array
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake", "apricot fritters");

// returns 5
echo sizeof($desserts);

// create array
$movies = array("romance" => "Moulin Rouge", "epic" =>
"Gladiator",
"action" => "Rambo", "horror" => "The Sixth Sense");

// returns 4
echo sizeof($movies);
?>
```

If you're using a hash, the array_keys() and array_values() functions come in handy to get a list of all the keys and values within the array.

```
<?
// create array
$starwars = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke", "bad guy" => "Darth", "worse guy" => "The Emperor");

// returns the array ("princess", "teacher", "new hope", "bad
guy", "worse
guy")
array_keys($starwars);

// returns the array ("Leia", "Yoda", "Luke", "Darth", "The
Emperor")
array_values($starwars);
?>
```

And the in_array() function can tell you whether or not a particular value exists in an array.

```
<?
// create array
$starwars = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke", "bad guy" => "Darth", "worse guy" => "The Emperor");

// returns true
echo in_array("Yoda", $starwars);

// returns false
echo in_array("Obi-Wan", $starwars);
?>
```

An alternative (and sometimes more efficient) approach to the one above would be to scan the array for a value match and return the corresponding key. If this is what you need, consider the array_search() function, which is designed just for this purpose:

```
<?
// create array
$starwars = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke", "bad guy" => "Darth", "worse guy" => "The Emperor");

// returns "bad guy"
echo array_search("Darth", $starwars);
?>
```

You can convert array elements into regular PHP variables with the list() and extract() functions. The list() function assigns array elements to variables,

```
<?
// create array
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake", "apricot fritters");

// assign elements to variables
list($a, $b, $c, $d, $e) = $desserts;

// returns "tiramisu"
echo $b;
?>
```

while the extract() function iterates through a hash, converting the key−value pairs into corresponding variable−value pairs.

```
<?
// create array
$starwars = array("princess" => "Leia", "teacher" => "Yoda");

// assign elements to variables
extract($starwars);

// returns "Leia"
echo $princess;
?>
```

Push And Pull

You can add an element to the end of an existing array with the array_push() function,

```
<?
// create array
$superheroes = array("spiderman", "superman", "captain
marvel", "green
lantern");

array_push($superheroes, "the incredible hulk");

// $superheroes now contains ("spiderman", "superman",
"captain marvel",
"green lantern", "the incredible hulk")
?>
```

and remove an element from the end with the interestingly−named array_pop() function.

```
<?
// create array
$superheroes = array("spiderman", "superman", "captain
marvel", "green
lantern");

array_pop($superheroes);

// $superheroes now contains ("spiderman", "superman",
"captain marvel")
?>
```

If you need to pop an element off the top of the array, you can use the array_shift() function,

```
<?
// create array
$superheroes = array("spiderman", "superman", "captain
marvel", "green
lantern");

array_shift($superheroes);

// $superheroes now contains ("superman", "captain marvel",
"green lantern")
```

```
?>
```

while the array_unshift() function takes care of adding elements to the beginning of the array.

```
<?
// create array
$superheroes = array("spiderman", "superman", "captain
marvel", "green
lantern");

array_unshift($superheroes, "the human torch");

// $superheroes now contains ("the human torch", "spiderman",
"superman",
"captain marvel", "green lantern")
?>
```

In case you need an array of a specific length, you can use the array_pad() function to create an array and pad it with empty elements (or elements containing a user−defined value). The following example should make this clearer:

```
<?
// create array
$students = array();

// returns an array containing 4 null values
$students = array_pad($students, 4, "");
?>
```

The second argument also specifies the direction of padding, while the third argument to the function specifies the value to be used for the empty elements.

```
<?
// create array
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
"chocolate
fudge cake", "apricot fritters");

// returns an array containing 8 elements
// ("chocolate mousse", "tiramisu", "apple pie", "chocolate
fudge cake",
"apricot fritters", "dummy", "dummy", "dummy")
// array is padded to the right since the size is positive
```

```
$desserts = array_pad($desserts, 8, "dummy");

// this would return the same array, but padded to the left
(note the
negative size)
// ("dummy", "dummy", "dummy", "chocolate mousse", "tiramisu",
"apple pie",
"chocolate fudge cake", "apricot fritters")
$desserts = array_pad($desserts, -8, "dummy");
?>
```

Slice And Dice

PHP allows you to extract a subsection of the array with the array_slice() function, in much the same way that the substr() function allows you to extract a section of a string. Here's what it looks like:

```
array_slice(array, start, length)
```

where "array" is an array variable, "start" is the index to begin slicing at, and "length" is the number of elements to return from "start".

Here's an example:

```
<?
// create array
$colours = array("red", "green", "blue", "yellow");

// returns the array ("green", "blue")
$slice = array_slice($colours, 1, 2);
?>
```

You can also use a negative index for the "start" position, to force PHP to begin counting from the right instead of the left.

```
<?
// create array
$colours = array("red", "green", "blue", "yellow");

// returns the array ("green", "blue")
$slice = array_slice($colours, -3, 2);
?>
```

The array_splice() function allows you to splice one or more values into an existing array. Here's what it looks like:

```
array_splice(array, start, length, replacement-values)
```

where "array" is an array variable, "start" is the index to begin slicing at, "length" is the number of elements to return from "start", and "replacement-values" are the values to splice in.

Here's an example:

```
<?
// create array
$trio = array("huey", "dewey", "louie");

// section to be spliced in
$splice = array("larry", "curly", "moe");

// $trio now contains ("huey", "larry", "curly", "moe")
array_splice($trio, 1, 2, $splice);
?>
```

Where Am I?

A number of built−in functions are available for use while iterating through an array − you'll typically use these in combination with a "while" or "for" loop.

The current() function returns the currently−in−use element of an array (beginning with the first element),

```
<?
// create array
$friends = array("Rachel", "Monica", "Phoebe", "Joey",
"Chandler", "Ross");

// returns "Rachel"
echo current($friends);
?>
```

while the key() function returns the corresponding array key.

```
<?
// create array
$friends = array("Rachel", "Monica", "Phoebe", "Joey",
"Chandler", "Ross");

// returns "Rachel"
echo current($friends);

// returns 0
echo key($friends);
?>
```

The next() and prev() functions move forward and backward through the array.

```
<?
// create array
$friends = array("Rachel", "Monica", "Phoebe", "Joey",
"Chandler", "Ross");

// returns "Rachel"
echo current($friends);

// move forward
next($friends);
```

```
// returns "Monica"
echo current($friends);

// move forward
next($friends);

// returns "Phoebe"
echo current($friends);

// move backwards
prev($friends);

// returns "Monica"
echo current($friends);
?>
```

The reset() and end() functions move to the beginning and end of an array respectively,

```
<?
// create array
$friends = array("Rachel", "Monica", "Phoebe", "Joey",
"Chandler", "Ross");

// returns "Rachel"
echo current($friends);

// move to end
end($friends);

// returns "Ross"
echo current($friends);

// move forward
reset($friends);

// returns "Rachel"
echo current($friends);
?>
```

while the each() function comes in handy when you need to iterate through an array.

```
<?
// create array
$music = array("pop", "rock", "jazz", "blues");
```

```
// creates the array ("0" => 0, "1" => "pop", "key" => 0,
"value" => "pop")
$this = each($music);

// returns "0" - no key, as this is not a hash
echo $this["0"];

// returns "pop"
echo $this["1"];

// returns "0" - no key, as this is not a hash
echo $this["key"];

// returns "pop"
echo $this["value"];
?>
```

Confused? Don't be - every time each() runs on an array, it creates a hash containing four keys: "0", "1", "key" and "value". The "0" and "key" keys contain the name of the currently-in-use key of the array (or hold the value 0 if the array does not contain keys), while the "1" and "value" keys contain the corresponding value.

Here's another example to make this clearer:

```
<?
// create array
$music = array("pop" => "Britney Spears", "rock" =>
"Aerosmith", "jazz" =>
"Louis Armstrong");

// creates the array ("0" => "pop", "1" => "Britney Spears",
"key" =>
"pop", "value" => "Britney Spears")
$this = each($music);

// returns "pop"
echo $this["0"];

// returns "Britney Spears"
echo $this["1"];

// returns "pop"
echo $this["key"];

// returns "Britney Spears"
echo $this["value"];
```

```
// move forward
// creates the array ("0" => "rock", "1" => "Aerosmith", "key"
=> "rock",
"value" => "Aerosmith")
$this = each($music);
?>
```

Sorting Things Out

If you need to, you can rearrange the elements within an array with PHP's numerous sorting functions. The simplest of these is the array_reverse() function, which merely reverses the order of elements within an array.

```
<?
// create array
$trio = array("huey", "dewey", "louie");

// returns ("louie", "dewey", "huey")
array_reverse($trio);
?>
```

The shuffle() function randomly reshuffles the elements within an array,

```
<?
// create array
$trio = array("huey", "dewey", "louie");

// shuffles elements of array to return ("dewey", "louie",
"huey")
shuffle($trio);
?>
```

while the array_unique() function helps you strip out duplicate values from an array.

```
<?
// create array
$clones = array("Tom", "Tom", "Harry", "Tom", "Harry",
"Harry", "Harry",
"Tom");

// returns ("Tom", "Harry")
array_unique($clones);
?>
```

The sort() function can be used to sort an array alphabetically or numerically,

```
<?
// create array
$animals = array("antelope", "zebra", "skunk", "baboon",
```

```
"viper");

// returns ("antelope", "baboon", "skunk", "viper", "zebra")
sort($animals);
?>
```

while the rsort() function does the same thing (just the other way around).

```
<?
// create array
$animals = array("antelope", "zebra", "skunk", "baboon",
"viper");

// returns ("zebra", "viper", "skunk", "baboon", "antelope")
rsort($animals);
?>
```

The ksort() function sorts a hash by key (you can reverse the sort order with the krsort() function),

```
<?
// create array
$numbers = array("2" => "duet", "13" => "baker's dozen", "-15"
=>
"temperature", "3" => "stooges");

// returns ("-15" => "temperature", "2" => "duet", "3" =>
"stooges", "13"
=> "baker's dozen")
ksort($numbers);

// returns ("13" => "baker's dozen", "3" => "stooges", "2" =>
"duet", "-15"
=> "temperature")
krsort($numbers);
?>
```

The usort() function lets you apply your own sort function to the elements of an array. The function that you define must be capable of comparing two values, and must return a positive, negative or zero value depending on whether the first value being compared is greater than, less than or equal to the second value.

An example might help to make this clearer. The following code snippet defines a custom sort function, which arranges elements according to their length.

```
<?
// compare two values on length
function check_length($str1, $str2)
{
if (strlen($str1) > strlen($str2))
{
return 1;
}
elseif (strlen($str1) == strlen($str2))
{
return 0;
}
else
{
return −1;
}
}

// create array
$animals = array("antelope", "zebra", "skunk", "baboon",
"viper", "yak");

// returns ("yak", "skunk", "viper", "zebra", "baboon",
"antelope")
usort($animals, "check_length");
?>
```

In a similar manner, you can apply a user−defined comparison function to the keys of a hash with the uksort() function − I'll leave this to you to experiment with.

The natsort() function makes it possible to sort array elements the way a human − rather than a computer − would. Consider the following example:

```
<?
// create array
$mixed = array(1, "zebra", "skunk", "baboon", 13, "viper",
−99, "yak");

// returns ("−99", "baboon", "skunk", "viper", "yak", "zebra",
"1", "13")
sort($mixed);

// returns ("−99", "1", "13", "baboon", "skunk", "viper",
"yak", "zebra")
natsort($mixed);
?>
```

Flipping Out

A number of other functions are available to help you perform miscellaneous array manipulation. For example, the array_flip() function can be used to interchange keys and values in an array,

```
<?
// create array
$music = array("pop" => "Britney Spears", "rock" =>
"Aerosmith", "jazz" =>
"Louis Armstrong");

// returns a new array ("Britney Spears" => "pop", "Aerosmith"
=> "rock",
"Louis Armstrong" => "jazz")
array_flip($music);
?>
```

while the array_merge() function combines multiple arrays into one single array.

```
<?
// create arrays
$good_guys = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke");
$bad_guys = array("bad guy" => "Darth", "worse guy" => "The
Emperor");

// returns a combined array("princess => Leia", "teacher =>
Yoda", "new
hope => Luke", "bad guy => Darth", "worse guy => The Emperor")
$starwars = array_merge($good_guys, $bad_guys);
?>
```

The array_diff() function accepts two or more arrays as arguments, and returns an array containing all those values of the first array which are absent in the remaining ones,

```
<?
// create arrays
$subset = array(7, 14, 21, 28);
$list = array(1, 4, 7, 8, 0, 23, 45, 15, 67, 29, 22);

// returns ("14", "21", "28")
array_diff($subset, $list);
```

```
?>
```

while the array_intersect() function does the opposite, calculating a list of those values of the first array which are present in all the remaining ones.

```
<?
// create arrays
$subset = array(7, 14, 21, 28);
$list = array(1, 4, 7, 8, 0, 23, 45, 15, 67, 29, 22);

// returns ("7")
array_intersect($subset, $list);
?>
```

The array_sum() function adds all the elements of an array and returns the total,

```
<?
// create an array
$subset = array(7, 14, 21, 28);

// returns 70
echo array_sum($subset);
?>
```

while the array_count_values() function calculates the frequency with which values appear within an array.

```
<?
// create array
$clones = array("Tom", "Tom", "Harry", "Tom", "Harry",
"Harry", "Harry",
"Tom", "Frank");

// returns the hash ("Tom => 4", "Harry => 4", "Frank => 1")
array_count_values($clones);
?>
```

The array_rand() function randomly returns one or more keys from an array.

```
<?
// create array
$desserts = array("chocolate mousse", "tiramisu", "apple pie",
```

```
"chocolate
fudge cake", "apricot fritters");

// returns the random array (1, 3)
array_rand($desserts, 2);

// create array
$starwars = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke", "bad guy" => "Darth", "worse guy" => "The Emperor");

// returns the random array ("princess", "bad guy")
array_rand($starwars, 2);
?>
```

And finally, the very useful array_walk() function allows you to run a user−defined function on every element of an array. The following example uses it to apply a specific number format to all the numbers in an array:

```
<?
// create arrays
$numbers = array(1, 567, 1.6777777777777, 0.031, 100.1,
−98.6);
$new_numbers = array();

// function to format numbers and add them to new array
function format($num)
{
global $new_numbers;
$new_numbers[] = sprintf("%1.2f", $num);
}

// runs the function format() on every element of $numbers
array_walk($numbers, "format");

// $new_numbers now contains ("1.00", "567.00", "1.68",
"0.03", "100.10",
"−98.60")
?>
```

And that's about all I have. I hope you enjoyed this article, and that it offered you some insight into the massive amount of power at your disposal when it comes to manipulating arrays. Should you require more information, the best place to go is the PHP manual page on arrays at http://www.php.net/manual/en/ref.array.php

Take care, and I'll see you soon!

Note: All examples in this article have been tested on Linux/i586 with PHP 4.0.6. Examples are illustrative only, and are not meant for a production environment. YMMV!