

Creating an Array

To define an array variable, name it using standard PHP variable naming rules and populate it with elements using the `array()` function, as illustrated in the following:

```
<?php
// define an array
$flavors = array('strawberry', 'grape', 'vanilla', 'caramel', 'chocolate');
?>
```

An alternative way to define an array is by specifying values for each element using index notation, like this:

```
<?php
// define an array
$flavors[0] = 'strawberry';
$flavors[1] = 'grape';
$flavors[2] = 'vanilla';
$flavors[3] = 'caramel';
$flavors[4] = 'chocolate';
?>
```

To create an associative array, use keys instead of numeric indices:

```
<?php
// define an associative array
$fruits['red'] = 'apple';
$fruits['yellow'] = 'banana';
$fruits['purple'] = 'plum';
$fruits['green'] = 'grape';
?>
```

Modifying Array Elements

To add an element to an array, assign a value using the next available index

number or key:

```
<?php
// add an element to a numeric array
$flavors[5] = 'mango';
// if you don't know the next available index
// this will also work
$flavors[] = 'mango';
// add an element to an associative array
$fruits['pink'] = 'peach';
?>
```

To modify an element of an array, assign a new value to the corresponding scalar variable. If you wanted to replace the flavor “strawberry” with “blueberry” in the \$flavors array created previously, you’d use the following:

```
<?php
// modify an array
$flavors[0] = 'blueberry';
?>
```

Processing Arrays with Loops

To iteratively process the data in a PHP array, loop over it using any of the loop constructs discussed in Chapter 4. To better understand this, create and run the following script:

```
<html>
<head></head>
<body>
Today's shopping list:
<ul>
<?php
// define array
$shoppingList = array('eye of newt', '
wing of bat', 'tail of frog');
// loop over it
// print array elements
for ($x = 0; $x < sizeof($shoppingList); $x++)
{
echo "<li>$shoppingList[$x]";
}
?>
</ul>
</body>
</html>
```

The foreach() Loop

While on the topic of arrays and loops, it is worthwhile to spend a few minutes discussing the new loop type introduced in PHP 4.0 for the purpose of iterating over an array: the foreach() loop. This loop runs once for each element of TEAM the array, moving forward through the array on each iteration. On each run, the statements within the curly braces are executed, and the currently selected array element is made available through a temporary loop variable. Unlike a for() loop, a foreach() loop doesn’t need a counter or a call to sizeof(); it keeps track of its position in the array automatically.

To better understand how this works, rewrite the previous example using the

foreach() loop:

```
<html>
<head></head>
<body>
Today's shopping list:
<ul>
<?php
// define array
$shoppingList = array('eye of newt', 'wing of bat', ↵
'tail of frog');
// loop over it
foreach ($shoppingList as $item)
{
echo "<li>$item";
}
?>
</ul>
</body>
</html>
```

You can process an associative array with a foreach() loop as well, although the manner in which the temporary variable is constructed is a little different to accommodate the key-value pairs. Try the following script to see how this works:

```
<html>
<head></head>
<body>
I can see:
<ul>
<?php

// define associative array
$animals = array ('dog' => 'Topsy', 'cat' => 'Tabitha', ↵
'parrot' => 'Polly');
// iterate over it
foreach ($animals as $key => $value)
{
echo "<li>a $key named $value";
}
?>
</ul>
</body>
```

```
</html>
```

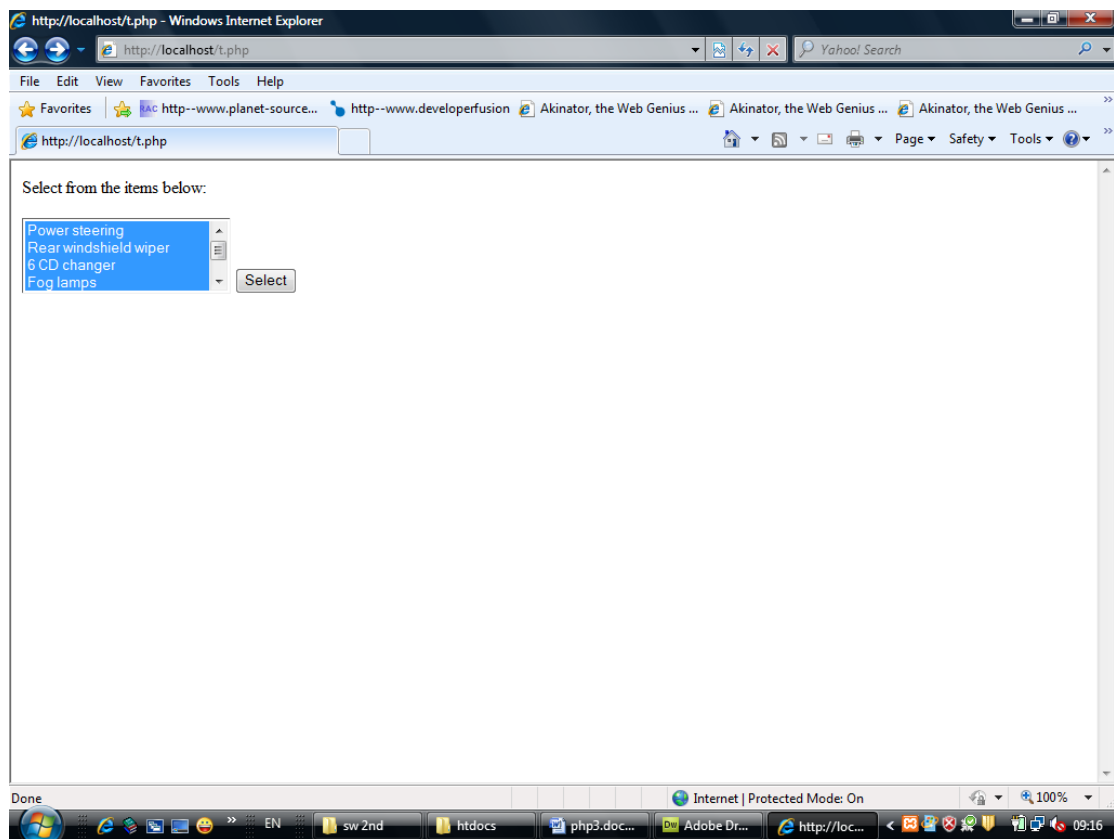
Grouping Form Selections with Arrays

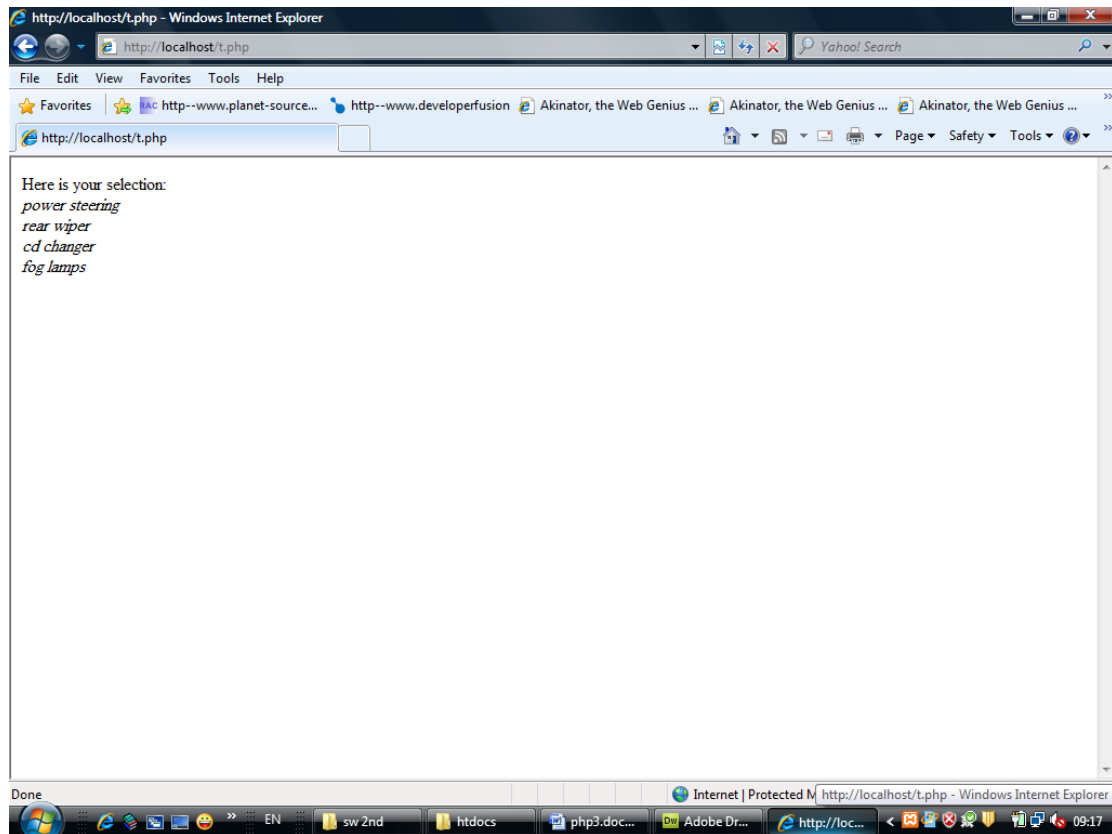
In addition to their obvious uses, arrays and loops also come in handy when processing forms in PHP. For example, if you have a group of related checkboxes or a multiselect list, you can use an array to capture all the selected form values in a single variable for greater ease in processing. To see how this works, create and run the following script:

```
<html>
<head></head>
<body>
<?php
// check if form has been submitted
if (!$_POST['submit'])
{
// if not, display form
?>
Select from the items below: <br />
<form action="<?php $_SERVER['PHP_SELF']?>" method="POST">
<select name="options[]" multiple>
<option value="power steering">Power steering</option>
<option value="rear wiper">Rear windshield wiper</option>
<option value="cd changer">6 CD changer</option>
<option value="fog lamps">Fog lamps</option>
<option value="central locking">Central locking</option>
<option value="onboard navigation">Computer-based
navigation</option>
</select>
<input type="submit" name="submit" value="Select">
</form>
<?php
}
else
{
// form has been submitted
// check if any items were selected
// if so, display them
if (is_array($_POST['options']))
{
echo 'Here is your selection: <br />';
// use a foreach() loop to read and display array elements
foreach ($_POST['options'] as $o)
{
```

```
echo "<i>$o</i><br />";
}
}
else
{
echo 'Nothing selected';
}
}
?>

</body>
</html>
```





Using Array Functions

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

The following example illustrates this:

```
<?php
// define an array
$menu = array('breakfast' => 'bacon and eggs', 'lunch' => 'roast beef',
'dinner' => 'lasagna');
// returns the array ('breakfast', 'lunch', 'dinner')
$result = array_keys($menu);
// returns the array ('bacon and eggs', 'roast beef', 'lasagna')
$result = array_values($menu);
?>
```

To check if a variable is an array, use the `is_array()` function, as in the following:

```
<?php
// create array
$desserts = array('chocolate mousse', 'tiramisu');
// returns 1 (true)
echo is_array($desserts);
?>
```

You can convert array elements into regular PHP variables with the `list()` and `extract()` functions. The `list()` function assigns array elements to variables, as in the following example:

```
<?php
// define an array
$flavors = array('strawberry', 'grape', 'vanilla');
// extract values into variables
list ($flavor1, $flavor2, $flavor3) = $flavors;
// returns "strawberry"
echo $flavor1;
?>
```

The `extract()` function iterates through a hash, converting the key-value pairs into corresponding variable-value pairs. Here's how:

```
<?php
// define associative array
$fruits = array('red' => 'apple', 'yellow' => 'banana', 'purple' => 'plum', 'green' => 'grape');
// extract values into variables
extract ($fruits);
// returns "banana"
echo $yellow;
?>
```

You can add an element to the end of an existing array with the `array_push()` function, and remove an element from the end with the interestingly named `array_pop()` function. If you need to pop an element off the top of the array, you can use the `array_shift()` function, while the `array_unshift()` function takes care of adding elements to the beginning of the array. The following example demonstrates all these functions:

```
<?php
// define array
$students = array('Tom', 'Jill', 'Harry');
// remove an element from the beginning
array_shift($students);
// remove an element from the end
array_pop($students);
// add an element to the end
array_push($students, 'John');
// add an element to the beginning
```

```
array_unshift($students, 'Ronald');
// array now looks like ('Ronald', 'Jill', 'John')
print_r($students);
?>
```

The `explode()` function splits a string into smaller components on the basis of a user-specified pattern, and then returns these elements as an array. This function is particularly handy if you need to take a string containing a list of items (for example, a comma-delimited list) and separate each element of the list for further processing. Here's an example:

```
<?php
// define string
$string = 'English Latin Greek Spanish';
// split on whitespace
$languages = explode(' ', $string);
// $languages now contains ('English', 'Latin', 'Greek', 'Spanish')
?>
```

Obviously, you can also do the reverse: the `implode()` function creates a single string from all the elements of an array, joining them together with a userdefined

separator. Revising the previous example, you have the following:

```
<?php
// define string
$string = 'English Latin Greek Spanish';
// split on whitespace
$languages = explode(' ', $string);
// create new string
// returns "English and Latin and Greek and Spanish"
$newString = implode(" and ", $languages);
?>
```