Date and Time processing with PHP

By The Disenchanted Developer

Table of Contents

The Devil Is In The Details

Some once famously said, "the more things change, the more they remain the same". And nowhere is this more true than in the area of programming languages.

No matter what your poison may be − Perl, PHP, Python, JSP − every language comes with certain fundamental constructs and concepts that are widely−understood and used. For example, though they may be called by different names, almost every language comes with string, numeric and array variables, conditional tests, loops, functions, exception handlers et al. This commonality makes it possible for experienced developers to easily switch from one development environment to another. This switch is not necessarily instant − the devil is still in the details − but a sound understanding of basic programming principles can go a long way to make it more painless.

What does this have to do with anything? Well, almost every language − including PHP, which is the subject of this article − comes with certain basic capabilities for date and time value manipulation. These capabilities may be implemented as functions, objects or object methods, but they do exist, in more or less similar form, in almost every language. And, over the course of this article, I'll be looking at how they work, and how they can be used to add something new to your bag of PHP tricks.

Getting A Date

First up, before we get into anything too complicated, let's take a quick tour of the important date and time manipulation functions that come with PHP 4.1.

The simplest and most basic thing you'll want to do with PHP's date API is, obviously, get the current date and time. This is best accomplished via the getdate() function, which returns an associative array containing date and time information.

Here's an example:

```
<?
// get current date information as associative array
$current = getdate();

// print it
print_r($current);
?>
```

Here's what the output would look like:

```
Array
(
[seconds] => 7
[minutes] => 19
[hours] => 9
[mday] => 20
[wday] => 3
[mon] => 2
[year] => 2002
[yday] => 50
[weekday] => Wednesday
[month] => February
[0] => 1014176947
)
```

This information can easily be manipulated and massaged into whatever format you like. For example,

```
<?
// get current date
$current = getdate();

// turn it into strings
```

```
$current_time = $current["hours"] . ":" . $current["minutes"]
. ":" .
$current["seconds"]; $current_date = $current["mday"] . "." .
$current["mon"] . "." . $current["year"];

// print it
// this would generate output of the form "It is now 9:14:34
on
23.5.2001" echo "It is now $current_time on $current_date"; ?>
```

You can also give getdate() a timestamp, and have it convert that timestamp into an array for you. Consider the following example, which sends getdate() a UNIX timestamp for the date December 25 2001, and gets the information back as a much easier−to−read array:

```
<?
print_r(getdate(1009218600));
?>
```

The output is:

```
Array
(
[seconds] => 0
[minutes] => 0
[hours] => 0
[mday] => 25
[wday] => 2
[mon] => 12
[year] => 2001
[yday] => 358
[weekday] => Tuesday
[month] => December
[0] => 1009218600
)
```

How did I create the timestamp? Flip the page to find out.

A Stamp In Time...

Most of PHP's date functions work on the basis of timestamps. This timestamp is a unique numeric representation of a particular date, calculated as the number of seconds between January 1 1970 and the date and time specified, and makes it easier to perform arbitrary calculations on date and time values.

In PHP, UNIX timestamps are created via the mktime() function, which accepts a series of date and time parameters, and converts them into a timestamp. Here's an example:

```
<?
// get a timestamp for 14:35:20 April 1 2002
echo mktime(14, 35, 20, 4, 1, 2002);
?>
```

In this case, I've passed the mktime() function six parameters: the hour, minute, second, month, day and year for which I need a timestamp. The result is a long integer like this:

```
1017651920
```

This integer can now be passed to getdate() − or any other date function that accepts timestamps − and converted into a human−readable representation.

You can obtain a timestamp for the current moment in time by calling mktime() with no arguments:

```
<?
// get a timestamp for now
echo mktime();
?>
```

Notice also that the "0" key of the array returned by getdate() contains a UNIX timestamp representation of the date returned.

Race Against Time

If you don't like mktime(), you can also use the time() function, which returns the current timestamp (note that you cannot use this function to generate arbitrary timestamps, as is possible with mktime()).

If you need greater accuracy in your timestamp, take a look at the microtime() function, which returns a timestamp in seconds, together with an additional microsecond component. These two values can be added together to obtain a more precise timestamp value. Like time(), this function too cannot be used with arbitrary dates and times − it only returns a timestamp for the current instant.

The microtime() function is particularly useful when you need to time events accurately, or when you need an arbitrary, unique number to seed PHP's random number generator. Here's an example of the former:

```
<?

// function to calculate timestamp in microseconds
function tscalc()
{
// split returned timestamp
$arr = explode(" ", microtime());
// add microseconds to seconds
// to generate more precise timestamp
return $arr[0] + $arr[1];
}

// start time
$start = tscalc();

// do something
// over here, run a loop
$count = 0;
$x = 0;
while ($count < 500)
{
$x = $x + $count;
$count++;
}

// end time
$end = tscalc();

echo "Started at $start";
echo "Ended at $end";
echo "Total processing time was " . ($end−$start);
?>
```

And here's an example of the latter:

```
<?

// function to calculate timestamp in microseconds
function tscalc()
{
// split returned timestamp
$arr = explode(" ", microtime());
// add microseconds to seconds
// to generate more precise timestamp
return $arr[0] + $arr[1];
}

// seed random number generator
srand(tscalc());

// get a random number
echo rand();
?>
```

When Looks Do Matter

Once you've got yourself a timestamp, you can use the date() function to format it and make it look pretty. This date() function is easily one of the most useful functions in this collection − it allows you to massage that long, ugly timestamp into something that's a pleasure to read.

Here's an example:

```
<?
// format timestamp with date()
echo "It is now " . date("h:i d M Y", mktime());
?>
```

The output of this would be:

```
It is now 12:20 20 Feb 2002
```

The date() function accepts two arguments: a format string and a timestamp. This format string is a sequence of characters, each of which has a special meaning. Here's a quick list:

```
CHARACTER WHAT IT MEANS
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

d day of the month (numeric)

D day of the week (string)

F month (string)

m month (numeric)

Y year

h hour (in 12−hour format)

H hour (in 24−hour format)

a AM or PM

i minute

s second
```

This is just a brief list, take a look at the PHP manual for a complete list.

Using these special characters, it's possible to format a timestamp to display just the information you want. For example,

```
<?
// returns "12:28 pm 20 Feb 2002"
echo date("h:i a d M Y", mktime());

// returns "12:28 20 February 2002"
echo date("H:i d F Y", mktime());

// returns "02−20−2002"
echo date("m−d−Y", mktime());
?>
```

You can use date() in combination with mktime() to generate human−readable strings for any arbitrary date value. A common example of this involves using date() on MySQL DATETIME values, which are typically in the format

```
yyyy−mm−yy hh:mm:ss
```

Here's an example of how a MySQL value could be converted into a human−readable date value:

```
<?

// format MySQL DATETIME value into a more readable string
function
formatDate($val) {
$arr = explode("−", $val);
return date("d M Y", mktime(0,0,0, $arr[1], $arr[2],
$arr[0]));
}

// assume I got the value "2001−02−01 17:29:25" from a query
// this is too much information
// use formatDate() to clean it up
echo formatDate("2001−02−01 17:29:25")

// the output would be "01 Feb 2001"
?>
```

In the event that you want to add your own printable characters to the string, you might need to use single quotes around the format string and escape them with a backslash so that PHP does not get confused. Consider the following example, which illustrates the difference:

```
<?
// would generate "028 2345 2o3 12:23 20 Feb 2002"
echo date("It is now h:i d M Y", mktime());

// would generate "It is now 12:23 20 Feb 2002"
echo date('\I\t\ \i\s \n\o\w\ h:i d M Y', mktime());
?>
```

You might also like to take a look at the fairly cool strtotime() function, which can be used to convert any natural−language date or time statement into a UNIX timestamp. Here are a few examples:

```
<?
// returns "20 Feb 2002"
echo date("d M Y", strtotime("now"));

// returns "21 Feb 2002"
echo date("d M Y", strtotime("tomorrow"));

// returns "27 Feb 2002"
echo date("d M Y", strtotime("next week"));

// returns "05:00 14 Feb 2002"
echo date("h:i d M Y", strtotime("5 pm 6 days ago"));
?>
```

Checking Up

Options to getdate() include the localtime() and gettimeofday() functions, both of which return arrays containing time information. Take a look:

```
<?
// get current time as array
$current = gettimeofday();

// print it
print_r($current);
?>
```

This would generate the following output:

```
Array
(
[sec] => 1014189678
[usec] => 900172
[minuteswest] => -19800
[dsttime] => 0
)
```

Alternatively, you can use the localtime() function, which calls the system's localtime() function and returns the output as an array containing information similar to that returned by getdate(). Take a look:

```
<?
// get current time as array
// the second parameter tells PHP to create an associative
array // omit
it to create a regular integer-indexed array $current =
localtime(mktime(), TRUE);

// print it
print_r($current);
?>
```

Here's the output:

```
Array
(
[tm_sec] => 51
[tm_min] => 54
[tm_hour] => 12
[tm_mday] => 20
[tm_mon] => 1
[tm_year] => 102
[tm_wday] => 3
[tm_yday] => 50
[tm_isdst] => 0
)
```

It's unlikely that you'll ever use these, since most of what they provide is already available via the functions discussed previously. However, they're included here in case you ever have a need for this specialized information.

Finally, the checkdate() function tells you whether a given date is valid or not. Consider the following examples:

```
<?
// 31−Feb−2002 − false
echo checkdate(02, 31, 2002);

// 25−Dec−1956 − true
echo checkdate(12, 25, 1956);

// 31−Jun−2002 − false
echo checkdate(06, 31, 2002);
?>
```

This function comes in particularly handy if you need to validate date information entered into an online form − simply run the datestamp via checkdate() to see whether or not it's valid.

Back to Class

Now, how about doing something useful with all this information? This next example uses everything you've learned to construct a simple PHP class that prints a monthly calendar. Take a look:

```
<?
class Calendar
{
//
// class variables
//

// list of names for days and months
var $days = array("Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday");
var $months = array("", "January", "February", "March",
"April",
"May", "June", "July", "August", "September", "October",
"November",
"December");

// number of days in each month
var $totalDays = array(0, 31, 28, 31, 30, 31, 30, 31, 31, 30,
31, 30, 31);

// variables to hold current month, day and year
var $currYear;
var $currMonth;
var $currDay;

//
// class methods
//

// constructor
function Calendar($year, $month)
{

// current values
$this->currYear = $year;
$this->currMonth = $month;
$this->currDay = date("j");

// if leap year, modify $totalDays array appropriately
if (date("L", mktime(0, 0, 0, $this->currMonth, 1,
$this->currYear)))
{
```

```
$this->totalDays[2] = 29;
}

}

// this prints the HTML code to display the calendar
function display()
{
// find out which day the first of the month falls on
$firstDayOfMonth = date("w", mktime(0, 0, 0,
$this->currMonth, 1, $this->currYear));

// start printing table
echo "<table border=0 cellpadding=2 cellspacing=5>\n";

// header
echo "<tr>\n";
echo "<td colspan=7 align=center><font face=Arial
size=-1><b>" . $this->months[$this->currMonth] . " " .
$this->currYear .
"</b></font></td>\n";
echo "</tr>\n";

// day names
echo "<tr>\n";
for ($x=0; $x<7; $x++)
{
echo "<td><font face=Arial size=-2>" .
substr($this->days[$x],0,3) . "</font></td>\n";
}
echo "</tr>\n";

// start printing dates
echo "<tr>\n";

// display blank spaces until the first day of the month
for ($x=1; $x<=$firstDayOfMonth; $x++)
{
// this comes in handy to find the end of each
7-day block
$rowCount++;
echo "<td><font face=Arial
size=-2>&nbsp;</font></td>\n";
}

// counter to track the current date
$dayCount=1;
while ($dayCount <= $this->totalDays[$this->currMonth])
{
```

```
// use this to find out when the 7-day block is
complete and display a new row
if ($rowCount % 7 == 0)
{
echo "</tr>\n<tr>\n";
}

// print date
// if today, display in different colour
if ($dayCount == date("j") && $this->currYear ==
date("Y") && $this->currMonth == date("n"))
{
echo "<td align=center
bgcolor=Silver><font face=Arial size=-1>$dayCount</font>";
}
else
{
echo "<td align=center><font face=Arial
size=-1>$dayCount</font>";
}

echo "</td>\n";
// increment counters
$dayCount++;
$rowCount++;
}
echo "</tr>\n";
echo "</table>\n";
}

// end of class
}
?>
```

Let's see how this works.

The first thing to do is set a few class variables to hold the month and year to be displayed − these variables will be used throughout the class, and are crucial to it functioning correctly.

```
<?
// list of names for days and months
var $days = array("Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday");
var $months = array("", "January", "February", "March",
"April",
"May", "June", "July", "August", "September", "October",
"November",
"December");

// number of days in each month
var $totalDays = array(0, 31, 28, 31, 30, 31, 30, 31, 31, 30,
31, 30, 31);

// variables to hold current month, day and year
var $currYear;
var $currMonth;
var $currDay;
?>
```

The values of the current year and month are set in the constructor, and are obtained via user input when an instance of the class is created.

```
<?
// current values
$this->currYear = $year;
$this->currMonth = $month;
$this->currDay = date("j");

?>
```

The date() function even lets you find out if the year under consideration is a leap year − if it is, it's necessary to modify the $totalDays array for the month of February. Since the date() function only works on a correctly−formatted UNIX timestamp, the mktime() function is used to first convert the numeric month and year into an acceptable format.

```
<?
// if leap year, modify $totalDays array appropriately
if (date("L", mktime(0, 0, 0, $this->currMonth, 1,
$this->currYear)))
{
$this->totalDays[2] = 29;
}
?>
```

Turning The Tables

Now, let's get into the display() method, which takes care of actually printing the calendar. There's one important thing needed here: the day of the week on which the first of the month falls.

```
<?
// find out which day the first of the month falls on
$firstDayOfMonth = date("w", mktime(0, 0, 0,
$this−>currMonth, 1, $this−>currYear));
?>
```

The first day of the month (from the $firstDayOfMonth variable) and the last day (from the $totalDays array) provide the bounding values for the month view I'm going to be building.

I'll explain how this view is constructed row by row. The first row contains just the name of the current month.

```
<?
// header
echo "<tr>\n";
echo "<td colspan=7 align=center><font face=Arial
size=−1><b>" . $this−>months[$this−>currMonth] . " " .
$this−>currYear .
"</b></font></td>\n";
echo "</tr>\n";
?>
```

The next row contains seven cells, one for each day of the week − I've used the substr() function to display the first three letters of each day name from the $days array.

```
<?
// day names
echo "<tr>\n";
for ($x=0; $x<7; $x++)
{
echo "<td><font face=Arial size=−2>" .
substr($this−>days[$x],0,3) . "</font></td>\n";
}
echo "</tr>\n";
?>
```

The next few rows are all generated automatically. The first order of business is to place the first day of the month on the corresponding day. Since I already have a $firstDayOfMonth variable, I've used a simple loop to fill all the cells prior to that day with non−breaking spaces.

```
<?
// start printing dates
echo "<tr>\n";

// display blank spaces until the first day of the month
for ($x=1; $x<=$firstDayOfMonth; $x++)
{
// this comes in handy to find the end of each
7−day block
$rowCount++;
echo "<td><font face=Arial
size=−2>&nbsp;</font></td>\n";
}
?>
```

The $rowCount variable is simultaneously keeping track of the number of slots (cells) being filled up − I'll use this a little further down to determine when the end of the week has been reached.

Once the first day of the month is determined, another "for" loop (iterating from 1 to the number of days in the month) is used to generate the remaining rows and cells of the table. The $rowCount and $dayCount variables are incremented at each stage, and the $rowCount variable is divided by 7 to find out when the seven slots available in each row are filled up.

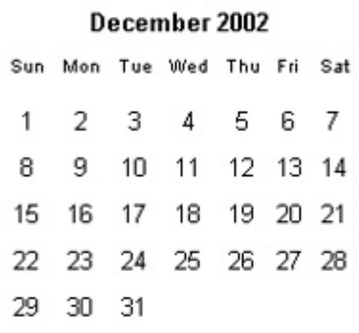
```
<?
// counter to track the current date
$dayCount=1;
while ($dayCount <= $this−>totalDays[$this−>currMonth])
{
// use this to find out when the 7−day block is
complete and display a new row
if ($rowCount % 7 == 0)
{
echo "</tr>\n<tr>\n";
}

// print date
// if today, display in different colour
if ($dayCount == date("j") && $this−>currYear ==
date("Y") && $this−>currMonth == date("n"))
{
echo "<td align=center
bgcolor=Silver><font face=Arial size=−1>$dayCount</font>";
}
else
{
echo "<td align=center><font face=Arial
size=-1>$dayCount</font>";
}

echo "</td>\n";
// increment counters
$dayCount++;
$rowCount++;
}
echo "</tr>\n";
echo "</table>\n";
}
?>
```

Notice that I've inserted an "if" statement into the loop to display the current date in a different colour, if a match is found.

Here's an example of how this class might be used:

```
<?
include("Calendar.inc");
$cal = new Calendar(2002, 12);
$cal->display();
?>
```

And here's what it would look like:



December 2002

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Pretty cool, huh?

This is just one example of the myriad uses to which PHP's date functions can be put. There are many more; I'll leave them to your imagination. Until next time...be good! Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.20 and PHP 4.1.1. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!