How to use timestamps to work with dates

A *timestamp* uses an integer to represent a date and time. This integer stores the number of seconds since midnight on January 1, 1970 GMT (Greenwich Mean Time). This point in time is known as the *Unix Epoch*.

Today, most systems store a timestamp as a 32-bit signed integer. As a result, timestamps on these systems can range from December 13, 1901 to January 19, 2038. When these systems reach the upper limit in 2038, the number of seconds "rolls over" to the lower limit in 1901. When this happens, this will cause bugs due to erroneous calculations, and it's known as the *year 2038 problem*. Since this problem is similar to the Y2K problem that affected many applications in the year 2000, the year 2038 problem is also known as the *Y2K38 problem*. One way to solve this problem is to use DateTime objects as described later in this chapter instead of using timestamps.

How to create and format timestamps

Figure 10-1 shows how to use the date function to create and format timestamps. In the first argument for this function, each format code represents one part of a date or time. For example, the 'm' character represents the month part of a date. Note, however, that many parts of a date or time have multiple representations. For example, the month can be represented as a number without a leading zero, as a number with a leading zero, as a three-letter abbreviation, or as multiple letters that spell out the full name.

The first six statements show how this works. Here, the first three statements format the current date three different ways. Then, the next two statements format the current time in two different ways. Finally, the sixth statement formats both the date and the time.

Any character in the format string that isn't a format code is displayed as itself. For example, the first five statements include slashes, dashes, commas, spaces, and colons in the format string. However, if you want to display a format code as itself in the format string, you must code a backslash before the format code. For example, the sixth statement uses the backslash to include the 'a' and 't' characters in the format string. This is necessary because both the 'a' and 't' character are format codes. In this figure, the 't' character isn't listed in the table of format codes, but it is a format code. For a complete list of format codes, look up the date function in the PHP documentation.

Although you usually use the date function to format the timestamp for the current date and time, you can also use it to format other timestamps. To do that, you supply the timestamp as the second parameter as shown in the seventh statement. Here, the timestamp is an integer value for March 15, 2012. In the next figure, though, you'll learn several other ways to create timestamps and store them in variables. Then, you can use the date function to format those variables.

The date function

Name	Description
`date($format[, $ts])`	Returns a string formatted as a date as specified by the format string. By default, this function works with the current date and time. However, you can use the second parameter to specify a timestamp for any date or time.

Common format codes for the date function

Character	Description	From	To
`D`	Day of week – three letters	Mon	Sun
`l (lowercase L)`	Day of week – full name	Monday	Sunday
`n`	Month – no leading zero	1	12
`m`	Month – leading zero	01	12
`M`	Month – three letters	Jan	Dec
`F`	Month – full name	January	December
`j`	Day of month – no leading zero	1	31
`d`	Day of month – leading zero	01	31
`Y`	Year – four digits	2010	
`L`	Leap year (1) or not (0)	0	1
`g`	Hours – 12-hour format, no leading zero	1	12
`h`	Hours – 24-hour format, no leading zero	0	23
`G`	Hours – 12-hour format, leading zero	01	12
`H`	Hours – 24-hour format, leading zero	00	23
`i`	Minutes – leading zero	00	59
`s`	Seconds – leading zero	00	59
`a`	am/pm – lowercase	am	pm
`A`	AM/PM – uppercase	AM	PM
`T`	Time zone abbreviation	EST	
`U`	Seconds since Unix epoch	-2,147,483,648	2,147,483,647

How to format a timestamp

```
$date1 = date('n/j/Y');              // 3/15/2011
$date2 = date('Y-m-d');              // 2011-03-15
$date3 = date('l, F d, Y');          // Monday, March 15, 2011
$date4 = date('g:i a');              // 1:30 pm
$date5 = date('H:i:s');              // 13:30:00
$date6 = date('Y-m-d \a\t H:i:s');   // 2011-03-15 at 13:30:00
$date7 = date('Y-m-d', 1331843400);  // 2012-03-15
```

Description

- A *timestamp* is an integer that represents a date and time as the number of seconds since midnight, January 1, 1970 GMT. This date and time is known as the *Unix epoch*.
- To include a literal value for a format character in the format string, code a backslash before the character.

Figure 10-1 How to create and format timestamps

How to work with timestamps

Figure 10-2 shows how to use four functions to work with timestamps. However, you should know that PHP provides many more functions to work with timestamps. For a complete list or for more information about the functions described in this figure, you can visit the URL shown at the top of this figure.

To create a timestamp, you can use either the time or mktime functions. The time function creates a timestamp for the current date and time, and the mktime function creates a timestamp from the values you provide. These functions return an integer value for the timestamp. To format the timestamp so it shows a date and time, you can use date function shown in the previous figure.

When using the mktime function, the parameters are optional. As a result, you can omit parameters starting with the last parameter and working towards the first parameter. If you omit a parameter, the mktime function uses the value from the current date and time. For example, the last parameter specifies the year. So, if you omit this parameter, the mktime function uses the current year for the timestamp. Similarly, if you omit the last two parameters, the mktime function uses the current year and day for the timestamp.

The mktime function doesn't validate the values provided to it. If a value for a part of the date or time is out of range, the other parts are adjusted to account for the excess amount in the out-of-range value. For example, if you provide a minute value of 90, this function adds 1 hour to the hour part and 30 minutes to the minute part.

To validate a date before using it, you can use the checkdate function. This is illustrated by the second example in this figure. Here, the checkdate function returns a FALSE value because the specified date is November 31, which isn't valid because November only has 30 days.

Although PHP doesn't provide a built-in function to validate a time, you can create a checktime function like the one in the third example to validate a 24 hour time. This function checks to make sure the specified hour is from 0 to 23 and the specified minutes and seconds are from 0 to 59. If so, it returns a true value. Otherwise, it returns a false value.

To get the parts of a timestamp, you can use the getdate function. This function takes a timestamp and returns an associative array that contains the date and time parts. In the example in this figure, the code uses the mktime function to make a new timestamp and the getdate function to convert that timestamp to an array named $parts. Then, it copies each part of the date from the $parts array into a separate variable.

URL for a list of all PHP timestamp functions

http://www.php.net/manual/en/ref.datetime.php

Functions for working with timestamps

Name	Description
`time()`	Returns the current date and time as a timestamp.
`mktime([$h[, $m[, $s[, $M[, $D[, $Y]]]]]])`	Returns a timestamp based on the time and date given. Any parts omitted are set to the value from the current date and time.
`checkdate($M, $D, $Y)`	Returns true if the provided month, day, and year values are a valid date.
`getdate([$ts])`	Returns an array containing the parts of the specified timestamp. If you omit the parameter, this function uses the current date and time.

How to create a timestamp

```
$now = time();                                // for example, 1265656521
$expires = mktime(13, 30, 0, 3, 15, 2012);   // 3/15/2012 13:30:00
$expires = mktime(13, 30, 0, 3, 15);         // uses current year
$expires = mktime(13, 30, 0, 3);             // uses current year and day
```

How to validate a date

```
$valid_date = checkdate(11, 31, 2012);       // returns FALSE
```

How to validate a time

A custom function for validating time

```
function checktime($h, $m, $s) {
    return $h >= 0 && $h < 24 && $m >= 0 && $m < 60 && $s >= 0 && $s < 60;
}
```

A statement that calls the custom function

```
$valid_time = checktime(12, 30, 0);          // TRUE
```

How to get the parts of a timestamp

```
$expires = mktime(13, 30, 0, 3, 15, 2012);
$parts = getdate($expires);
$year    = $parts['year'];      // 2012
$mon     = $parts['mon'];       // Month number - 3
$month   = $parts['month'];     // Month name - 'March'
$mday    = $parts['mday'];      // Day of month - 15
$weekday = $parts['weekday'];   // Weekday - 'Monday'
$wday    = $parts['wday'];      // Weekday as number - 1
$hours   = $parts['hours'];     // Hours - 13
$minutes = $parts['minutes'];   // Minutes - 30
$seconds = $parts['seconds'];   // Seconds - 0
```

Figure 10-2 How to work with timestamps

How to use the strtotime function

The strtotime function presented in figure 10-3 is one of the most versatile date and time functions in just about any programming language. Fortunately, most of the skills for using it also apply to DateTime objects. As a result, it makes sense to learn how to use this function whether you're going to use timestamps or DateTime objects.

You can use the strtotime function to generate a timestamp for most strings that specify dates. To do that, you can pass this function a string that specifies a date, a time, or a date and a time. This works for strings that store dates and times in most common formats. These types of strings are known as *absolute templates*.

In the first example, for instance, the first three statements specify strings for dates. Since these statements don't set the time, the time is set to midnight. Similarly, since the third statement doesn't set the year, the year is set to the current year. The next two statements specify strings for times. Since these statements don't set the date, the date is set to the current date. Finally, the last statement specifies a string for a date and a time.

The second example shows how to specify a format string that gets a timestamp that's relative to current date and time. The part of a format string that specifies a date or time relative to another date or time is known as a *relative template*. In the second example, for instance, the first statement uses a relative template to add one hour. The second statement subtracts two days. The third statement sets the date to the next day. Note that this resets the time to midnight. The fourth statement uses the relative template of "tomorrow" to set the date to the next day and also uses an absolute template to set the time to 10:15 am.

The fifth statement specifies next Sunday. The sixth statement sets the date to the last day of the current month. The seventh statement sets the date to the first day of the next month.

The eighth statement sets the date to third Wednesday of the current month. The ninth statement uses an absolute template to set the month to November, uses a relative template to set the day to the second Tuesday of that month, and uses an absolute template to set the time to 8 am.

The templates in the format string are processed from left to right. Most relative templates are cumulative and the changes are applied when the end of the format string is reached. Some relative templates reset the time to midnight. For these templates, you can specify time by coding an absolute time template after the relative template.

The third example shows how to use the strtotime function to modify a timestamp. To start, the first statement uses the mktime function to make a timestamp. Then, the second statement uses the strtotime function to get a timestamp that's three weeks later at 6 pm.

The strtotime function provides many more templates than we can cover in this chapter. Most of the time, these functions can convert your strings to timestamps. As a result, to get the most from this function, you may need to do some testing and experimentation.

The strtotime function

Name	Description
`strtotime($str[, $ts])`	Returns a timestamp for the specified string. By default, this function works relative to the current date and time. However, if you supply the $ts parameter, this function works relative to the specified timestamp. This function is able to parse most date and time formats and most relative time specifications.

Types of templates used in strtotime

Type	Description
Absolute	A format string that specifies a date or time. A date without a time sets the time to midnight. A partial date (for example, just month and day) leaves the unspecified date parts unchanged. A time without a date does not change the date. A partial time specification (for example, only hours and minutes) sets the unspecified time parts to zero.
Relative	An offset to be added to or subtracted from the base date and time. Some relative dates set the time to midnight.

How to generate a timestamp with an absolute template

```
// Examples assume a current time of Sun 04/08/2012 1:30:00 pm
$date1 = strtotime('2013-06-01');                 // Sat 06/01/2013 00:00
$date2 = strtotime('6/1/2013');                   // Sat 06/01/2013 00:00
$date3 = strtotime('Jun 1');                      // Fri 06/01/2012 00:00
$date4 = strtotime('8:45');                       // Sun 04/08/2012 08:45
$date5 = strtotime('8am');                        // Sun 04/08/2012 08:00
$date6 = strtotime('2013-02-29 8:45am');          // Fri 03/01/2013 08:45
```

How to generate a timestamp with a relative template

```
// Examples assume a current time of Sun 04/08/2012 1:30:00 pm
$date1 = strtotime('+1 hour');                    // Sun 04/08/2012 14:30
$date2 = strtotime('-2 days');                    // Sat 04/06/2012 13:30
$date3 = strtotime('tomorrow');                   // Mon 04/09/2012 00:00
$date4 = strtotime('tomorrow 10:15am');           // Mon 04/09/2012 10:15
$date5 = strtotime('next sunday');                // Sun 04/15/2012 00:00
$date6 = strtotime('last day of');                // Mon 04/30/2012 13:30
$date7 = strtotime('first day of next month');    // Tue 05/01/2012 13:30
$date8 = strtotime('third wednesday of');         // Wed 04/18/2012 00:00
$date9 = strtotime('nov second tue of 8am');      // Tue 11/13/2012 08:00
```

How to modify a timestamp

```
$checkout = mktime(13, 30, 0, 4, 8, 2012);
$due_date = strtotime('+3 weeks 6pm', $checkout);
```

Description

- An *absolute template* specifies a specific date, time, or both.
- A *relative template* specifies an offset to the base date and time. To specify an offset, you can use most English descriptions of time and date offsets.

Figure 10-3 How to use the strtotime function

Examples of working with timestamps

To round out your skills for working with timestamps, figure 10-4 shows four examples of working with timestamps. The first example determines if a year is a leap year. To start, it defines a function named is_leapyear that takes a timestamp as its parameter. Within the function, the statement uses the date function with a format string of 'L' to check if the timestamp is a leap year. If the date function returns a string of "1", the year is a leap year and the is_leapyear function returns a true value. Otherwise, this function returns a FALSE value.

The second example compares two timestamps to determine which comes first. Since timestamps are stored as integers, you can use relational operators such as the less than sign to compare timestamps. To start, the variable named $now stores a timestamp for the current date and time. Then, the variable named $exp stores a timestamp for the expiration date of a credit card. This expiration date is set to the first day of the next month after April 2012. If $exp is less than $now, the current date has passed the expiration date. In other words, the card is current if the current date and time is before midnight of May 1, 2012. Otherwise, the card is expired.

The third example displays a more detailed message about the card expiration date. To start, this code gets the current time and stores the expiration date in the typical mm/yyyy format. However, the strtotime function needs a month and year in the yyyy-mm format. So, the code uses the substr function to extract the month and year and changes the format of the expiration date.

After changing the format of the expiration date, the code uses the strtotime function to calculate the expiration date and time (midnight of May 1, 2012). Then, the code calculates the number of days between the current time and the expiration date by subtracting the two timestamps, dividing by the number of seconds in a day (86400), and using the floor function to truncate the result. Finally, this code checks the number of days and displays an appropriate message that includes the number of days between the current date and the expiration date.

The fourth example displays a countdown until the New Year. To start, this code gets the current time. Then, it uses the strtotime function to get midnight on Jan 1 of the next year. Next, the code calculates the number of seconds between the two dates.

After getting the number of seconds, this code calculates the number of days remaining and subtracts that many seconds from the total number of seconds. It repeats this type of calculation to find the number of hours, minutes, and seconds remaining. Finally, it displays the number of days, hours, minutes, and seconds remaining until the New Year.

If the current date is before daylight savings time and the time is 12 noon, this code displays 13 hours. That's because timestamps take the change from daylight savings time to standard time into account. For a date after daylight savings time and a time of 12 noon, this code displays 12 hours.

Determine if a year is a leap year

```
function is_leapyear($ts) {
    return (date('L', $ts) == '1');
}
$year_2010 = is_leapyear(strtotime('2010-1-1'));   // FALSE
$year_2012 = is_leapyear(strtotime('2012-1-1'));   // TRUE
```

Display a simple message about an expiration date

```
$now = time();
$exp = strtotime('2012-4 first day of next month midnight');
if ($exp < $now) {
    echo 'Your card has expired.';
} else {
    echo 'Your card has not expired.';
}
```

Display a detailed message about an expiration date

```
$now = time();
$exp = '04/2012';    // Typical expiration date format

// Change exp format from mm/yyyy to yyyy-mm
$month = substr($exp, 0, 2);
$year  = substr($exp, 3, 4);
$exp = $year . '-' . $month;

// Set expiration date and calculate the number of days from current date
$exp = strtotime($exp . ' first day of next month midnight');
$days = floor(($exp - $now) / 86400);   // There are 86400 seconds/day

// Display a message
if ($days < 0) {
    echo 'Your card expired ' . abs($days) . ' days ago.';
} else if ($days > 0) {
    echo 'Your card expires in ' . $days . ' days.';
} else {
    echo 'Your card expires at midnight.';
}
```

Display a countdown until the New Year

```
$now = time();
$new_year = strtotime('next year Jan 1st', $now);

// Calculate the days, hours, minutes, and seconds
$seconds = $new_year - $now;
$days = floor($seconds / 86400);
$seconds -= $days * 86400;
$hours = floor($seconds / 3600);
$seconds -= $hours * 3600;
$minutes = floor($seconds / 60);
$seconds -= $minutes * 60;

// Display the countdown
echo "$days days and $hours:$minutes:$seconds remaining to the New Year.";
```

Figure 10-4 Examples of working with timestamps