

# CHAPTER 3

---

## Working with Dates and Times

### IN THIS CHAPTER:

- 3.1 Getting the Current Date and Time
- 3.2 Formatting Timestamps
- 3.3 Checking Date Validity
- 3.4 Converting Strings to Timestamps
- 3.5 Checking for Leap Years
- 3.6 Finding the Number of Days in a Month
- 3.7 Finding the Day-in-Year or Week-in-Year  
Number for a Date
- 3.8 Finding the Number of Days  
or Weeks in a Year
- 3.9 Finding the Day Name for a Date
- 3.10 Finding the Year Quarter for a Date
- 3.11 Converting Local Time to GMT
- 3.12 Converting Between Different Time Zones
- 3.13 Converting Minutes to Hours
- 3.14 Converting Between PHP  
and MySQL Date Formats
- 3.15 Comparing Dates
- 3.16 Performing Date Arithmetic
- 3.17 Displaying a Monthly Calendar
- 3.18 Working with Extreme Date Values

Like most programming languages, PHP comes with a fairly full-featured set of functions for date and time manipulation. Two of these functions are probably familiar to you from your daily work as a developer—the `date()` function for formatting dates and times and the `mktime()` function for generating timestamps. But it’s unlikely that you’ve had as much contact with the other members of the collection—the `strtotime()` function, the `gmdate()` function, or the `microtime()` function.

These functions, together with many more, make it easy to solve some fairly vexing date/time manipulation problems. Over the course of this chapter, I’ll show you how to solve such problems, with listings for converting between time zones; checking the validity of a date; calculating the number of days in a month or year; displaying a monthly calendar; performing date arithmetic; and working with date values outside PHP’s limits.

### NOTE

---

*PHP’s date and time functions were rewritten in PHP 5.1, with the result that every date or time function expects the default time zone to be set (and generates a notice if this is not the case). The listings in this chapter assume that this default time zone has previously been set, either via the `$TZ` environment variable or the `date.timezone` setting in the `php.ini` configuration file. In the rare cases when it is necessary to over-ride the system-wide time zone setting, PHP offers the `date_default_timezone_set()` function, which can be invoked to set the time zone on a per-script basis, as may be seen in the listing in “3.12: Converting Between Different Time Zones.”*

---

## 3.1 Getting the Current Date and Time

### Problem

You want to display the current date and/or time.

### Solution

Use PHP’s `getdate()` function:

```
<?php
// get current date and time
$now = getdate();
```

```
// turn it into strings
$currentTime = $now["hours"] . ":" . $now["minutes"] . ":"
              . $now["seconds"];
$currentDate = $now["mday"] . "." . $now["mon"] . "." . $now["year"];

// result: "It is now 12:37:47 on 30.10.2006" (example)
echo "It is now $currentTime on $currentDate";
?>
```

## Comments

PHP's `getdate()` function returns an array of values representing different components of the current date and time. Here's an example of what the array might look like:

```
Array
(
    [seconds] => 34
    [minutes] => 14
    [hours] => 9
    [mday] => 23
    [wday] => 2
    [mon] => 5
    [year] => 2006
    [yday] => 137
    [weekday] => Monday
    [month] => February
    [0] => 1107752144
)
```

As the previous listing illustrates, it's easy enough to use this array to generate a human-readable date and time value. However, formatting options with `getdate()` are limited, so if you need to customize your date/time display extensively, look at the listing in “3.2: Formatting Timestamps” for an alternative way of accomplishing the same thing.

### **NOTE**

---

*Notice that the 0<sup>th</sup> element of the array returned by `getdate()` contains a UNIX timestamp representation of the date returned—the same one that `mktime()` would generate.*

---

## 3.2 Formatting Timestamps

### Problem

You want to turn a UNIX timestamp into a human-readable string.

### Solution

Use PHP's `date()` function to alter the appearance of the timestamp with various formatting codes:

```
<?php
// get date
// result: "30 Oct 2006" (example)
echo date("d M Y", mktime()) . " \n";

// get time
// result: "12:38:26 PM" (example)
echo date("h:i:s A", mktime()) . " \n";

// get date and time
// result: "Monday, 30 October 2006, 12:38:26 PM" (example)
echo date ("l, d F Y, h:i:s A", mktime()) . " \n";

// get time with timezone
// result: "12:38:26 PM UTC"
echo date ("h:i:s A T", mktime()) . " \n";

// get date and time in ISO8601 format
// result: "2006-10-30T12:38:26+00:00"
echo date ("c", mktime());
?>
```

### Comments

PHP's `date()` function is great for massaging UNIX timestamps into different formats. It accepts two arguments—a format string and a timestamp—and uses the format string to turn the timestamp into a human-readable value. Each character in the format string has a special meaning, and you can review the complete list at <http://www.php.net/date>.

The `date()` function is usually found in combination with the `mktime()` function, which produces a UNIX timestamp for a particular instant in time. This timestamp is represented as the number of seconds since January 1 1970 00:00:00 Greenwich Mean Time (GMT). Called without any arguments, `mktime()` returns a timestamp for the current instant in time; called with arguments, it returns a timestamp for the instant represented by its input. The following snippets illustrate this:

```
<?php
// get current timestamp
// result: 1162218979 (example)
echo mktime() . " \n";

// get timestamp for 01:00 AM 31 Jan 2007
// result: 1170205200
echo mktime(1,0,0,1,31,2007);
?>
```

### NOTE

---

*An alternative to the `mktime()` function is the `time()` function, which returns a UNIX timestamp for the current instant in time. Unlike `mktime()`, however, `time()` cannot be used to produce timestamps for arbitrary date values.*

---

## 3.3 Checking Date Validity

### Problem

You want to check if a particular date is valid.

### Solution

Use PHP's `checkdate()` function:

```
<?php
// check date 31-Apr-2006
// result: "Invalid date"
echo checkdate(31,4,2006) ? "Valid date" : "Invalid date";
?>
```

## Comments

Applications that accept date input from a user must validate this input before using it for calculations or date operations. The `checkdate()` function simplifies this task considerably. It accepts a series of three arguments, representing day, month and year, and returns a Boolean value indicating whether the combination make up a legal date.

An alternative way of accomplishing the same thing can be found in the PEAR Calendar class, available from <http://pear.php.net/package/Calendar>. This class offers an `isValid()` method to test the validity of a particular date value. The following listing illustrates this:

```
<?php
// include Calendar class
include "Calendar/Day.php";

// initialize Day object to 31-Apr-2006
$day = & new Calendar_Day(2006, 4, 31);

// check date
// result: "Invalid date"
echo $day->isValid() ? "Valid date" : "Invalid date";
?>
```

---

## 3.4 Converting Strings to Timestamps

### Problem

You want to convert a string, encapsulating a date or time value, into the corresponding UNIX timestamp.

### Solution

Use PHP's `strtotime()` function:

```
<?php
// define string
$str = "20030607";

// convert string to timestamp
$ts = strtotime($str);
```

```
// format as readable date/time value
// result: "Saturday, 07 June 2003 12:00:00 AM" (example)
echo ($ts === -1) ? "Invalid string" : date("l, d F Y h:i:s A", $ts);
?>
```

## Comments

PHP's `strtotime()` function performs the very important function of converting a human-readable date value into a UNIX timestamp, with minimal calculation required on the part of the application. The date value can be any English-language date descriptor; `strtotime()` will attempt to identify it and return the corresponding timestamp. If `strtotime()` cannot convert the description to a timestamp, it will return `-1`.

In addition to date strings, the `strtotime()` function also accepts English-language time descriptors like “now,” “next Wednesday,” or “last Friday,” and you can use it to perform rudimentary date arithmetic. The following listing illustrates this:

```
<?php
// assume now is "Monday, 30 October 2006 02:56:34 PM"

// define string
$str = "next Friday";

// convert string to timestamp
$ts = strtotime($str);

// format as readable date/time value
// result: "Friday, 03 November 2006 12:00:00 AM"
echo ($ts === false) ? "Invalid string" : date("l, d F Y h:i:s A", $ts);

// define string
$str = "2 weeks 6 hours ago";

// convert string to timestamp
$ts = strtotime($str);

// format as readable date/time value
// result: "Monday, 16 October 2006 08:56:34 AM"
echo ($ts === false) ? "Invalid string" : date("l, d F Y h:i:s A", $ts);
?>
```

**TIP**

*For more sophisticated date arithmetic, take a look at the listing in “3.16: Performing Date Arithmetic.” Read more `strtotime()` examples in the PHP manual at <http://www.php.net/strtotime>.*

---

## 3.5 Checking for Leap Years

### Problem

You want to check if a particular year is a leap year.

### Solution

Write a function to see if the year number is divisible by 4 or 400, but not 100:

```
<?php
// function to test if leap year
function testLeapYear($year) {
    $ret = (($year%400 == 0) || ($year%4 == 0 && $year%100 != 0)) ?
    true : false;
    return $ret;
}

// result: "Is a leap year"
echo testLeapYear(2004) ? "Is a leap year" : "Is not a leap year";

// result: "Is not a leap year"
echo testLeapYear(2001) ? "Is a leap year" : "Is not a leap year";
?>
```

### Comments

A year is a leap year if it is fully divisible by 400, or by 4 but not 100. The function `testLeapYear()` in the previous listing encapsulates this logic, using PHP’s `%` operator to check for divisibility, and returns a Boolean value indicating the result.

An alternative way to do this is to use the `checkdate()` function to test for the presence of an extra day in February of that year. The following listing illustrates this:

```
<?php
// function to test if leap year
function testLeapYear($year) {
    return checkdate(2, 29, $year);
}

// result: "Is a leap year"
echo testLeapYear(2004) ? "Is a leap year" : "Is not a leap year";

// result: "Is not a leap year"
echo testLeapYear(2001) ? "Is a leap year" : "Is not a leap year";
?>
```

---

## 3.6 Finding the Number of Days in a Month

### Problem

You want to find the number of days in a particular month.

### Solution

Use PHP's `date()` function with the "t" modifier:

```
<?php
// get timestamp for month and year Mar 2005
$ts = mktime(0,0,0,3,1,2005);

// find number of days in month
// result: 31
echo date("t", $ts);
?>
```

## Comments

Given a UNIX timestamp, the `date()` function's "t" modifier returns the number of days in the corresponding month. The return value will range from 28 to 31.

An alternative way of accomplishing the same thing is to use the PEAR Date class, available at <http://pear.php.net/package/Date>. Here, a `Date()` object is first initialized to a specific day, month, and year combination, and then the class' `getDaysInMonth()` method is used to retrieve the number of days in that month. The next listing illustrates this:

```
<?php
// include Date class
include "Date.php";

// initialize Date object to 1-Mar-2005
$dt = new Date();
$dt->setYear(2005);
$dt->setMonth(3);
$dt->setDay(1);

// get number of days in month
// result: 31
echo $dt->getDaysInMonth();
?>
```

---

## 3.7 Finding the Day-in-Year or Week-in-Year Number for a Date

### Problem

You want to find the day-in-year or week-in-year number for a particular date.

### Solution

Use PHP's `date()` function with the "z" or "W" modifier:

```
<?php
// get day of year for 01-Mar-2008
// result: 61
echo date("z", mktime(0,0,0,3,1,2008))+1;
```

```
// get week of year for 01-Mar-2008
// result: 09
echo date("W", mktime(0,0,0,3,1,2008));
?>
```

## Comments

Given a UNIX timestamp, the `date()` function's "z" modifier returns the day number in the year, while the "W" modifier returns the week number. Note that day numbers are indexed from 0, so it is necessary to add 1 to the final result to obtain the actual day number in the year. Also look at the listing in “3.8: Finding the Number of Days or Weeks in a Year” for another application of this technique.

Alternatively, you can use PEAR's Date class, available from <http://pear.php.net/package/Date>, to obtain the week number. Here, a `Date()` object is first initialized to a specific day, month, and year combination, and then the class' `getWeekOfYear()` method is used to retrieve the week number for that date.

```
<?php
// include Date class
include "Date.php";

// initialize Date object to 1-Mar-2008
$dt = new Date();
$dt->setYear(2008);
$dt->setMonth(3);
$dt->setDay(1);

// get week number in year
// result: 9
echo $dt->getWeekOfYear();
?>
```

---

## 3.8 Finding the Number of Days or Weeks in a Year

### Problem

You want to find the number of days or weeks in a particular year.

## Solution

Use PHP's `date()` function with the "z" or "W" modifiers:

```
<?php
// get total number of days in the year 2001
$numDays = date("z", mktime(0,0,0,12,31,2001))+1;

// get total number of weeks in the year 2001
$numWeeks = date("W", mktime(0,0,0,12,28,2001));

// result: "There are 365 days and 52 weeks in 2001."
echo "There are $numDays days and $numWeeks weeks in 2001.\n";
?>
```

## Comments

Given a UNIX timestamp, the `date()` function's "z" modifier returns the day number in the year, while the "W" modifier returns the week number. By passing a timestamp representation of the last day or last week of the year, it's possible to quickly find the total number of days or weeks in the year. Also look at the listing in "3.7: Finding the Day-in-Year or Week-in-Year Number for a Date" for another application of this technique.

Note that the value returned by the "z" modifier is indexed from 0, so it is necessary to add 1 to the final result to obtain the actual number of days in the year.

---

## 3.9 Finding the Day Name for a Date

### Problem

You want to find which day of the week a particular date falls on.

### Solution

Use PHP's `date()` function with the "l" modifier:

```
<?php
// get timestamp for date 04-Jun-2008
$ts = mktime(0,0,0,6,4,2008);
```

```
// get day of week
// result: "Wednesday"
echo date("l", $ts);
?>
```

## Comments

Given a timestamp representing a particular date, the `date()` function's "l" modifier returns the weekday name corresponding to that date. If you need a numeric value (0 = Sunday, 1 = Monday, ...) rather than a string, use the "w" modifier instead.

---

## 3.10 Finding the Year Quarter for a Date

### Problem

You want to find which quarter of the year a particular date falls in.

### Solution

Use PHP's `date()` function with the "m" modifier:

```
<?php
// get timestamp for date 04-Jun-2008
$ts = mktime(0,0,0,6,4,2008);

// get quarter
// result: 2
echo ceil(date("m", $ts)/3);
?>
```

## Comments

Given a timestamp representing a particular date, the `date()` function's 'm' modifier returns the month number (range 1–12) corresponding to that date. To obtain the corresponding year quarter, divide the month number by 3 and round it up to the nearest integer with the `ceil()` function.

An alternative way of accomplishing the same thing is to use the PEAR Date class, available from <http://pear.php.net/package/Date>. Here, a `Date()` object is first initialized to a specific day, month, and year combination, and then the

class' `getQuarterOfYear()` method is used to retrieve the year quarter for that month. The next listing illustrates this:

```
<?php
// include Date class
include "Date.php";

// initialize Date object
$dt = new Date();
$dt->setYear(2008);
$dt->setMonth(6);
$dt->setDay(6);

// get quarter
// result: 2
echo $dt->getQuarterOfYear();
?>
```

---

## 3.11 Converting Local Time to GMT

### Problem

You want to convert local time to Greenwich Mean Time (GMT).

### Solution

Use PHP's `gmdate()` function:

```
<?php
// convert current local time (IST) to GMT
// result: "15:06:25 30-Oct-06 GMT" (example)
echo gmdate("H:i:s d-M-y T") . "\n";

// convert specified local time (IST) to GMT
// result: "23:00:00 01-Feb-05 GMT" (example)
$ts = mktime(4,30,0,2,2,2005);
echo gmdate("H:i:s d-M-y T", $ts);
?>
```

## Comments

The `gmdate()` function formats and displays a timestamp in GMT. Like the `date()` function, it accepts a format string that can be used to control the final appearance of the date and time value. Conversion to GMT is performed automatically based on the time zone information returned by the operating system.

An alternative way of finding GMT time is to find the local time zone offset from GMT, and subtract that from the local time. This offset can be found by using the `date()` function's "Z" modifier, which returns, in seconds, the time difference between the current location and Greenwich. A negative sign attached to the offset indicates that the location is west of Greenwich.

The next listing illustrates this:

```
<?php
// convert current local time (IST) to GMT
// result: "15:07:56 30-Oct-06 GMT" (example)
echo date("H:i:s d-M-y", time()-date("Z")) . " GMT \n";

// convert specified local time (IST) to GMT
// result: "23:00:00 01-Feb-05 GMT"
$ts = mktime(4,30,0,2,2,2005);
echo date("H:i:s d-M-y", $ts-date("Z", $ts)) . " GMT";
?>
```

---

## 3.12 Converting Between Different Time Zones

### Problem

You want to obtain the local time in another time zone, given its GMT offset.

### Solution

Write a PHP function to calculate the time in the specified zone:

```
<?php
// function to get time
// for another time zone
// given a specific timestamp and hour offset from GMT
```

```

function getLocalTime($ts, $offset) {
    // performs conversion
    // returns UNIX timestamp
    return ($ts - date("Z", $ts)) + (3600 * $offset);
}

// get current local time in Singapore
// result: "00:11:26 31-10-06 SST"
echo date("H:i:s d-m-y", getLocalTime(mktime(), 8)) . " SST \n";

// get current local time in India
// result: "21:41:26 30-10-06 IST"
echo date("H:i:s d-m-y", getLocalTime(mktime(), +5.5)) . " IST \n";

// get current local time in USA (Eastern)
// result: "11:11:26 30-10-06 EST"
echo date("H:i:s d-m-y", getLocalTime(mktime(), -5)) . " EST \n";

// get current local time in USA (Pacific)
// result: "08:11:26 30-10-06 PST"
echo date("H:i:s d-m-y", getLocalTime(mktime(), -8)) . " PST \n";

// get time in GMT
// when it is 04:30 AM in India
// result: "23:00:00 01-02-05 GMT "
echo date("H:i:s d-m-y", getLocalTime(mktime(4,30,0,2,2,2005), 0)) . "\n"
" GMT \n";
?>

```

## Comments

Assume here that you're dealing with two time zones: Zone 1 and Zone 2. The user-defined function `getLocalTime()` accepts two arguments: a UNIX timestamp for Zone 1 and the time zone offset, in hours from GMT, for Zone 2. Because it's simpler to perform time zone calculations from GMT, the Zone 1 UNIX timestamp is first converted to GMT (see the listing in "3.12: Converting Between Different Time Zones" for more on this step) and then the stated hour offset is added to it to obtain a new UNIX timestamp for Zone 2 time. This timestamp can then be formatted for display with the `date()` function.

Note that given UNIX timestamps are represented in seconds, the hour offset passed to `getLocalTime()` must be multiplied by 3600 (the number of seconds

in 1 hour) before the offset calculation can be performed. Note also that if the hour offset passed to `getLocalTime()` is 0, GMT time will be returned.

If this is too complicated for you, you can also perform time zone conversions with the PEAR Date class, available from <http://pear.php.net/package/Date>. Here, a `Date()` object is initialized and its current time zone is set with the `setTZ()` method. The corresponding time in any other region of the world can then be obtained by invoking the `convertTZ()` method with the name of the region.

Take a look:

```
<?php
// include Date class
include "Date.php";

// initialize Date object
$d = new Date("2005-02-01 16:29:00");

// set time zone
$d->setTZ('Asia/Calcutta');

// convert to UTC
// result: "2005-02-01 10:59:00"
$d->toUTC();
echo $d->getDate() . " \n";

// convert to American time (EST)
// result: "2005-02-01 05:59:00"
$d->convertTZ(new Date_TimeZone('EST'));
echo $d->getDate() . " \n";

// convert to Singapore time
// result: "2005-02-01 18:59:00"
$d->convertTZ(new Date_TimeZone('Asia/Singapore'));
echo $d->getDate() . " \n";
?>
```

A complete list of valid region names for time zone conversion can be obtained from the package documentation.

**TIP**


---

*There's also a third "shortcut" solution to this problem: simply use the `date_default_timezone_set()` function to set the default time zone to the target city or time zone, and use the `date()` function to return the local time in that zone. Here's an example:*

```
<?php
// set default time zone to destination
// result: "00:11:26 31-10-06 SST"
date_default_timezone_set('Asia/Singapore');
echo date("H:i:s d-m-y") . " SST \n";

// set default time zone to destination
// result: "08:11:26 30-10-06 PST"
date_default_timezone_set('US/Pacific');
echo date("H:i:s d-m-y") . " PST \n";
?>
```

---

## 3.13 Converting Minutes to Hours

### Problem

You want to convert between mm and hh:mm formats.

### Solution

Divide or multiply by 60 and add the remainder:

```
<?php
// define number of minutes
$mm = 156;

// convert to hh:mm format
// result: "02h 36m"
echo sprintf("%02dh %02dm", floor($mm/60), $mm%60);
?>

<?php
// define hours and minutes
$hhmm = "02:36";
```

```
// convert to minutes
// result: "156 minutes"
$arr = explode(":", $hmm);
echo $arr[0]*60 + $arr[1] . " minutes";
?>
```

## Comments

Which is more easily understood: “105 minutes” or “1 hour, 45 minutes”? The previous listing takes care of performing this conversion between formats.

Given the total number of minutes, the number of hours can be obtained by dividing by 60, with the remainder representing the number of minutes. The `printf()` function takes care of sticking the two pieces together.

Given a string in hh:mm format, the `explode()` function splits it on the colon (:) separator, converts the first element from hours to minutes by multiplying it by 60, and then adds the second element to get the total number of minutes.

---

## 3.14 Converting Between PHP and MySQL Date Formats

### Problem

You want to convert a MySQL DATETIME/TIMESTAMP value to a UNIX timestamp suitable for use with PHP’s `date()` function, or vice versa.

### Solution

To convert a MySQL TIMESTAMP/DATETIME type to a UNIX timestamp, use PHP’s `strtotime()` function or MySQL’s `UNIX_TIMESTAMP()` function:

```
<?php
// run database query, retrieve MySQL timestamp
$conn = mysql_connect("localhost", "user", "pass") ↓
or die ("Unable to connect!");
$query = "SELECT NOW() AS tsField";
$result = mysql_query($query) ↓
or die ("Error in query: $query. " . mysql_error());
$row = mysql_fetch_object($result);
mysql_close($conn);
```

## 92 PHP Programming Solutions

```
// convert MySQL TIMESTAMP/DATETIME field
// to UNIX timestamp with PHP strtotime() function
// format for display with date()
echo date("d M Y H:i:s", strtotime($row->tsField));
?>
```

```
<?php
// run database query, retrieve MySQL timestamp
// convert to UNIX timestamp using MySQL UNIX_TIMESTAMP() function
$conn = mysql_connect("localhost", "user", "pass")
or die ("Unable to connect!");
$query = "SELECT UNIX_TIMESTAMP(NOW()) as tsField";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
$row = mysql_fetch_object($result);
mysql_close($conn);

// timestamp is already in UNIX format
// so format for display with date()
echo date("d M Y H:i:s", $row->tsField);
?>
```

To convert a UNIX timestamp to MySQL's `TIMESTAMP/DATETIME` format, use the `date()` function with a custom format string, or use MySQL's `FROM_UNIXTIME()` function:

```
<?php
// create UNIX timestamp with mktime()
$ts = mktime(22,4,32,7,2,2007);

// turn UNIX timestamp into MySQL TIMESTAMP/DATETIME format (string)
// result: "2007-07-02 22:04:32"
echo date("Y-m-d H:i:s", $ts);

// turn UNIX timestamp into MySQL TIMESTAMP/DATETIME format (numeric)
// result: 20070702220432
echo date("YmdHis", $ts);
?>

<?php
// create UNIX timestamp with PHP mktime() function
$ts = mktime(22,4,32,7,2,2007);

// turn UNIX timestamp into MySQL TIMESTAMP/DATETIME format
// using MySQL's FROM_UNIXTIME() function
$conn = mysql_connect("localhost", "user", "pass")
or die ("Unable to connect!");
```

```

$query = "SELECT FROM_UNIXTIME('$ts') AS tsField";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
$row = mysql_fetch_object($result);
mysql_close($connection);
// result: "2007-07-02 22:04:32"
echo $row->tsField;
?>

```

## Comments

A common grouse of PHP/MySQL developers is the incompatibility between the date formats used by the two applications. Most of PHP's date/time functions use a UNIX timestamp; MySQL's DATETIME and TIMESTAMP fields only accept values in either YYYYMMDDHHMMSS or "YYYY-MM-DD HH:MM:SS" format. PHP's date() function will not correctly read a native DATETIME or TIMESTAMP value, and MySQL will simply zero out native UNIX timestamps. Consequently, converting between the two formats is a fairly important task for a PHP/MySQL developer.

Fortunately, there are a couple of ways to go about this, depending on whether you'd prefer to do the conversion at the PHP application layer or the MySQL database layer.

- ▶ At the PHP layer, you can convert a MySQL DATETIME or TIMESTAMP value into a UNIX timestamp by passing it to the PHP strtotime() function, which is designed specifically to parse and attempt to convert English-readable date values into UNIX timestamps (see the listing in “3.4: Converting Strings to Timestamps”). Going the other way, you can insert a UNIX timestamp into a MySQL DATETIME or TIMESTAMP field by first formatting it with the PHP date() function.
- ▶ At the MySQL layer, you can convert a MySQL DATETIME or TIMESTAMP value into a UNIX timestamp with the MySQL UNIX\_TIMESTAMP() function. Or, you can save a UNIX timestamp directly to a MySQL DATETIME or TIMESTAMP field by using MySQL's built-in FROM\_UNIXTIME() function to convert the timestamp into MySQL-compliant format.

---

## 3.15 Comparing Dates

### Problem

You want to compare two dates to see which is more recent.

## Solution

Use PHP's comparison operators to compare the timestamps corresponding to the two dates:

```
<?php
// create timestamps for two dates
$date1 = mktime(0,0,0,2,1,2007);
$date2 = mktime(1,0,0,2,1,2007);

// compare timestamps
// to see which represents an earlier date
if ($date1 > $date2) {
    $str = date ("d-M-Y H:i:s", $date2) . " comes before " .
date ("d-M-Y H:i:s", $date1);
} else if ($date2 > $date1) {
    $str = date ("d-M-Y H:i:s", $date1) . " comes before " .
date ("d-M-Y H:i:s", $date2);
} else {
    $str = "Dates are equal";
}
// result: "01-Feb-2007 00:00:00 comes before 01-Feb-2007 01:00:00"
echo $str;
?>
```

## Comments

PHP's comparison operators work just as well on temporal values as they do on numbers and strings. This is illustrated in the previous listing, which compares two dates to see which one precedes the other.

An alternative is the PEAR Date class, available from <http://pear.php.net/package/Date>. Comparing dates with this class is fairly simple: initialize two `Date()` objects, and then call the `compare()` method to see which one comes first. The `compare()` method returns 0 if both dates are equal, -1 if the first date is before the second, and 1 if the second date is before the first. Here's an illustration:

```
<?php
// include Date class
include "Date.php";

// initialize two Date objects
$date1 = new Date("2007-02-01 00:00:00");
$date2 = new Date("2007-02-01 01:00:00");
```

```
// compare dates
// returns 0 if the dates are equal
//      -1 if $date1 is before $date2
//      1 if $date1 is after $date2
// result: -1
echo Date::compare($date1, $date2);
?>
```

You could also use either one of the `Date()` objects' `before()` and `after()` methods on the other. The next listing illustrates this:

```
<?php
// include Date class
include "Date.php";

// initialize two Date objects
$date1 = new Date("2007-02-01 00:00:00");
$date2 = new Date("2006-02-01 00:00:00");

// check if $date1 is before $date2
// result: "false"
echo $date1->before($date2) ? "true" : "false";

// check if $date2 is before $date1
// result: "true"
echo $date1->after($date2) ? "true" : "false";
?>
```

### **TIP**

---

*You can compare a date relative to "today" with the `isPast()` and `isFuture()` methods. Look in the package documentation for examples.*

---

## 3.16 Performing Date Arithmetic

### Problem

You want to add (subtract) time intervals to (from) a date.

## Solution

Convert the date to a UNIX timestamp, express the time interval in seconds, and add (subtract) the interval to (from) the timestamp:

```
<?php
// set base date
$dateStr = "2008-09-01 00:00:00";

// convert base date to UNIX timestamp
// expressed in seconds
$timestamp = strtotime($dateStr);

// express "28 days, 5 hours, 25 minutes and 11 seconds"
// in seconds
$intSecs = 11 + (25*60) + (5*60*60) + (28*24*60*60);

// add interval (in seconds)
// to timestamp (in seconds)
// format result for display
// returns "2008-09-29 05:25:11"
$newDateStr = date("Y-m-d h:i:s", $timestamp + $intSecs);
echo $newDateStr;
?>
```

## Comments

When you're dealing with temporal data, one of the more common (and complex) tasks involves performing addition and subtraction operations on date and time values. Consider, for example, the simple task of calculating a date 91 days hence. Usually, in order to do this with any degree of precision, you need to factor in a number of different variables: the month you're in, the number of days in that month, the number of days in the months following, whether or not the current year is a leap year, and so on.

PHP doesn't provide built-in functions for this type of arithmetic, but it's nevertheless fairly easy to do. The previous listing illustrates one approach to the problem, wherein the time interval is converted to seconds and added to (or subtracted from) the base timestamp, also expressed in seconds.

Another option is to use PEAR's Date class, available from <http://pear.php.net/package/Date>. This class comes with two methods to perform date and time arithmetic: `addSpan()` and `subtractSpan()`. The "span" in both cases is a `DateSpan()` object, created from a delimited string containing day, hour, minute, and second intervals. This span is added to (or subtracted from) a previously

initialized `Date()` object, and a new date and time is calculated and returned as another `Date()` object. Here's an example:

```
<?php
// include Date class
include "Date.php";

// initialize Date object
$d = new Date("2007-02-01 00:00:00");

// add 28 days, 5 hours, 25 minutes and 11 seconds
// result: "2007-03-01 05:25:11"
$d->addSpan(new Date_Span("28:05:25:11"));
echo $d->getDate() . " \n";

// now subtract 1 day, 30 minutes
// result: "2007-02-28 04:55:11"
$d->subtractSpan(new Date_Span("01:00:30:00"));
echo $d->getDate();
?>
```

---

## 3.17 Displaying a Monthly Calendar

### Problem

You want to print a calendar for a particular month.

### Solution

Use PEAR's Calendar class:

```
<?php
// include Calendar class
include "Calendar/Month/Weekdays.php";
include "Calendar/Day.php";

// initialize calendar object
$month = new Calendar_Month_Weekdays(2008, 1);

// build child objects (days of the month)
$month->build();
```

```

// format as table
echo "<pre>";

// print month and year on first line
echo "          " . sprintf("%02d", $month->thisMonth()) . "/" .
$month->thisYear() . "\n";

// print day names on second line
echo " M T W T F S S\n";

// iterate over day collection
while ($day = $month->fetch()) {
    if ($day->isEmpty()) {
        echo "    ";
    } else {
        echo sprintf("%3d", $day->thisDay()) . " ";
    }

    if ($day->isLast()) {
        echo "\n";
    }
}
echo "</pre>";
?>

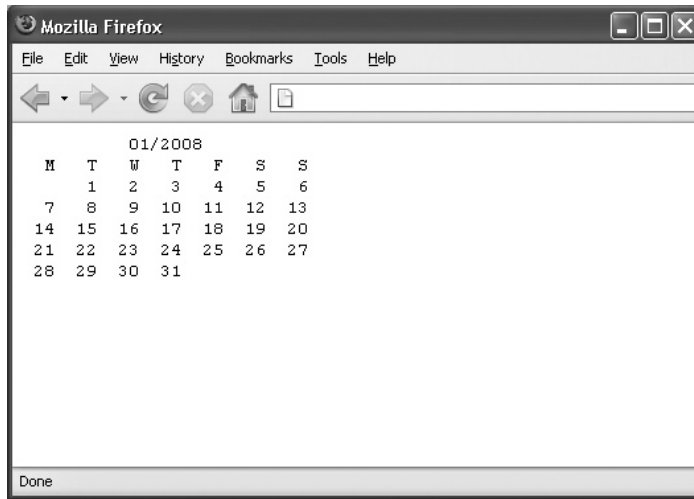
```

## Comments

Displaying a dynamic calendar on a Web page might seem trivial, but if you've ever tried coding it firsthand, you'll know the reality is somewhat different. Better than working your way through the numerous calculations and adjustments, then, is using the PEAR Calendar class, available from <http://pear.php.net/package/Calendar>. This class is designed specifically to generate a monthly or yearly calendar that you can massage into whatever format you desire.

The Calendar package includes a number of different classes, each for a specific purpose. The previous listing uses the `Calendar_Month_Weekdays()` class, which provides the methods needed to generate a monthly calendar sorted into weeks. (This is the same type you probably have hanging on your wall.) The class is initialized with a month and year, and its `build()` method is invoked to build the calendar data structure. A `while()` loop is then used in combination with the `fetch()` method to iterate over the Calendar data structure and print each day. Four utility method—`isFirst()`, `isLast()`, `isEmpty()` and `isSelected()`—enable you to customize the appearance of particular dates in the month.

Figure 3-1 illustrates the output of the listing above.



**Figure 3-1** A calendar generated with the PEAR Calendar class

The Calendar package is fairly sophisticated, and enables a developer to create and customize a variety of different calendar types. There isn't enough space here to discuss it in detail, so you should take a look at the examples provided with the package to understand what you can do with it.

## 3.18 Working with Extreme Date Values

### Problem

You want to work with dates outside the range 01-01-1970 to 19-01-2038.

### Solution

Use the ADOdb Date Library:

```
<?php
// include ADOdb date library
include "adodb-time.inc.php";
```

```
// get date representation for 01-Mar-1890
// returns "01-Mar-1890"
echo adodb_date("d-M-Y", adodb_mktime(4,31,56,3,1,1890)) . " \n";

// get date representation for 11-Jul-3690 10:31 AM
// result: "11-Jul-3690 10:31:09 AM"
echo adodb_gmtime("d-M-Y h:i:s A", adodb_mktime(16,1,9,07,11,3690)) . "
\n";

// get date representation for 11-Jul-3690 04:01 PM
// result: "11-Jul-3690 04:01:09 PM"
echo adodb_gmtime("d-M-Y h:i:s A", adodb_gmmktime(16,1,9,07,11,3690));
?>
```

## Comments

Because PHP uses 32-bit signed integers to represent timestamps, the valid range of a PHP timestamp is usually 1901–2038 on UNIX, and 1970–2038 on Windows. None of the built-in PHP date functions will work with dates outside this range. Needless to say, this is a Bad Thing.

You can work around this problem with the Active Data Objects Data Base (ADOdb) Date Library, a free PHP library that uses 64-bit floating-point numbers instead of 32-bit integers to represent timestamps, thus significantly increasing the valid range. This library is freely available from [http://phplens.com/phpeverywhere/adodb\\_date\\_library](http://phplens.com/phpeverywhere/adodb_date_library), and it provides 64-bit substitutes for PHP's native date and time functions, enabling you to work with dates from “100 A.D. to 3000 A.D. and later.”

As the previous listing illustrates, input and output parameters for the ADOdb functions are identical to those of the native PHP ones, enabling them to serve as drop-in replacements.