Most of the time, the date and time functionalities of PHP 5 are used for printing out the current date and time—to pretend that the web page is up to date (you would be surprised how many larger websites actually use this). But apart from that, working with date and time in PHP offers many other possibilities, most of which you will find in this chapter.

First, though, it seems appropriate to have a look at the PHP function that is probably used the most for working with dates—date(). This function can take the current date (or an arbitrary one) and extract some information about it, for example, the day, whether it’s a.m. or p.m., and what time it is according to the rather failed marketing stunt, “Swatch Internet Time.” To do so, you call date() and provide a string as the first parameter. This string may now contain a list of formatting symbols that can be seen in Table 3.1 (the PHP manual carries a list with more examples at http://php.net/date). Each of these symbols is replaced by the associated date/time value.
Table 3.1  Formatting Symbols for date()

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>am or pm</td>
</tr>
<tr>
<td>A</td>
<td>AM or PM</td>
</tr>
<tr>
<td>B</td>
<td>Swatch Internet Time (between 000 and 999)</td>
</tr>
<tr>
<td>c</td>
<td>Date in ISO 8601 format</td>
</tr>
<tr>
<td>d</td>
<td>Day of month (from 01 to 31)</td>
</tr>
<tr>
<td>D</td>
<td>Day of week (from Mon to Sun)</td>
</tr>
<tr>
<td>F</td>
<td>Month (from January to December)</td>
</tr>
<tr>
<td>g</td>
<td>Hour (from 1 to 12)</td>
</tr>
<tr>
<td>G</td>
<td>Hour (from 0 to 23)</td>
</tr>
<tr>
<td>h</td>
<td>Hour (from 01 to 12)</td>
</tr>
<tr>
<td>H</td>
<td>Hour (from 00 to 23)</td>
</tr>
<tr>
<td>i</td>
<td>Minutes (from 00 to 59)</td>
</tr>
<tr>
<td>I</td>
<td>Whether date is in DST (1) or not (0)</td>
</tr>
<tr>
<td>j</td>
<td>Day of month (between 1 and 31)</td>
</tr>
<tr>
<td>l</td>
<td>Day of month (from Sunday to Saturday)</td>
</tr>
<tr>
<td>L</td>
<td>Whether date is in a leap year (1) or not (0)</td>
</tr>
<tr>
<td>m</td>
<td>Month (from 01 to 12)</td>
</tr>
<tr>
<td>M</td>
<td>Month (from Jan to Dec)</td>
</tr>
<tr>
<td>n</td>
<td>Month (from 1 to 12)</td>
</tr>
<tr>
<td>O</td>
<td>Difference to GMT (for example, +0100 for one hour ahead)</td>
</tr>
<tr>
<td>r</td>
<td>Date in RFC 2822 format</td>
</tr>
<tr>
<td>s</td>
<td>Seconds (from 00 to 59)</td>
</tr>
<tr>
<td>S</td>
<td>Ordinal suffix for the day of month (st, nr, td, th)</td>
</tr>
<tr>
<td>t</td>
<td>Number of days in the provided month (from 28 to 31)</td>
</tr>
<tr>
<td>T</td>
<td>Time zone of server (for example, CET)</td>
</tr>
</tbody>
</table>
Table 3.1  

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Epoche value (seconds since January 1st, 1970, Midnight GMT)</td>
</tr>
<tr>
<td>w</td>
<td>Day of week (from 0—Sunday, to 6—Saturday)</td>
</tr>
<tr>
<td>W</td>
<td>Week number (according to ISO 8601, from 1 to 53)</td>
</tr>
<tr>
<td>y</td>
<td>Year (two digits)</td>
</tr>
<tr>
<td>Y</td>
<td>Year (four digits)</td>
</tr>
<tr>
<td>z</td>
<td>Day of year (from 0 to 365)</td>
</tr>
<tr>
<td>Z</td>
<td>Time zone difference to UTC (in seconds)</td>
</tr>
</tbody>
</table>

NOTE
Almost all of the formatting symbols shown in Table 3.1 are available since PHP 3. There are only two exceptions: Using W for determining the week number of a date was added in PHP 4.1.0, and using c for retrieving the ISO 8601 representation for a date (for example, 2006-06-30T12:34:56+01:00) came in PHP 5.

The function `date()` is very powerful and offers a broad range of ways to use it. However, especially if you have localized content, you need some good phrases. In this chapter, you will find many of them. PHP’s date and time functions have their own section in the PHP manual. You can find more information about `date()` and friends at http://php.net/datetime.
Using Text Within `date()`

**Using Text Within `date()`**

```php
<?php
echo date('Today is the day of the month');
?>
```

Using Ordinary Characters in `date()` (date.php)

Imagine you want to output the current date (or a specific date) and use custom strings in it. The code could look like a mess if you are trying it like this:

```php
<?php
echo 'Today is the day of the month'. date('jS'). date('F');
?>
```

The output of this script is something like this, depending on the current date:

```
Today is the 3rd day of the month May
```

The behavior of `date()` is the following: All characters within the first parameter that bear a special meaning (for example, formatting symbols) get replaced by the appropriate values. All other characters, however, remain unchanged. If a character is escaped using the backslash character (`\`), it is returned verbatim. So, the code at the beginning of this phrase shows a new version of the code that only uses one call to `date()`.
Automatically Localizing Dates

NOTE
If you are using double quotes instead of single quotes, you might get into trouble when escaping certain characters within `date()`. In the previous example, escaping the `n` would be done with `\n`, which (within double quotes) gets replaced by the newline character.

Automatically Localizing Dates

```php
setlocale(LC_TIME, 'en_US');
echo strftime('In (American) English: %c<br>');

<?php
setlocale(LC_TIME, 'en_US');
echo strftime('In (American) English: %c<br>');
setlocale(LC_TIME, 'en_gb');
echo strftime('In (British) English: %c<br>');
setlocale(LC_TIME, 'de_DE');
echo strftime('Auf Deutsch: %c<br>');
setlocale(LC_TIME, 'fr_FR');
echo strftime('En Français: %c');
?>
```

Localizing Dates Using `strftime()` (`strftime.php`)
The PHP function `strftime()` formats a date/time value according to the system's locale, for example, to the web server's local settings. Generally, the language of the system is automatically used. However, this can be overridden using `setlocale()`.

The function `strftime()` expects a format string (as does `date()`) in which it accepts a large number of special symbols. Table 3.2 contains a full list.
Automatically Localizing Dates

### Table 3.2 Formatting Symbols for strftime()

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Day of week (abbreviated)</td>
</tr>
<tr>
<td>%A</td>
<td>Day of week</td>
</tr>
<tr>
<td>%b or %h</td>
<td>Month (abbreviated)</td>
</tr>
<tr>
<td>%B</td>
<td>Month</td>
</tr>
<tr>
<td>%c</td>
<td>Date and time in standard format</td>
</tr>
<tr>
<td>%C</td>
<td>Century</td>
</tr>
<tr>
<td>%d</td>
<td>Day of month (from 01 to 31)</td>
</tr>
<tr>
<td>%D</td>
<td>Date in abbreviated format (mm/dd/yy)</td>
</tr>
<tr>
<td>%e</td>
<td>Day of month as a two-character string (from '1' to '31')</td>
</tr>
<tr>
<td>%g</td>
<td>Year according to the week number, two digits</td>
</tr>
<tr>
<td>%G</td>
<td>Year according to the week number, four digits</td>
</tr>
<tr>
<td>%H</td>
<td>Hour (from 00 to 23)</td>
</tr>
<tr>
<td>%I</td>
<td>Hour (from 01 to 12)</td>
</tr>
<tr>
<td>%j</td>
<td>Day of year (from 001 to 366)</td>
</tr>
<tr>
<td>%m</td>
<td>Month (from 01 to 12)</td>
</tr>
<tr>
<td>%M</td>
<td>Minute (from 00 to 59)</td>
</tr>
<tr>
<td>%n</td>
<td>Newline (\n)</td>
</tr>
<tr>
<td>%p</td>
<td>am or pm (or local equivalent)</td>
</tr>
<tr>
<td>%r</td>
<td>Time using a.m./p.m. notation</td>
</tr>
<tr>
<td>%R</td>
<td>Time using 24 hours notation</td>
</tr>
<tr>
<td>%S</td>
<td>Second (from 00 to 59)</td>
</tr>
<tr>
<td>%t</td>
<td>Tab (\t)</td>
</tr>
<tr>
<td>%T</td>
<td>Time in hh:mm format</td>
</tr>
<tr>
<td>%u</td>
<td>Day of week (from 1—Monday—to 7—Sunday)</td>
</tr>
</tbody>
</table>
Table 3.2  Continued

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%U</td>
<td>Week number (Rule: The first Sunday is the first day of the first week.)</td>
</tr>
<tr>
<td>%V</td>
<td>Week number (Rule: The first week in the year with at least four days counts as week number 1.)</td>
</tr>
<tr>
<td>%w</td>
<td>Day of week (from 0—Sunday to 6—Saturday)</td>
</tr>
<tr>
<td>%W</td>
<td>Week number (Rule: The first Monday is the first day of the first week.)</td>
</tr>
<tr>
<td>%x</td>
<td>Date in standard format (without the time)</td>
</tr>
<tr>
<td>%X</td>
<td>Time in standard format (without the date)</td>
</tr>
<tr>
<td>%y</td>
<td>Year (two digits)</td>
</tr>
<tr>
<td>%Y</td>
<td>Year (four digits)</td>
</tr>
<tr>
<td>%z or %Z</td>
<td>Time zone</td>
</tr>
</tbody>
</table>

Whenever it says *standard format* in Table 3.2, the formatting symbol gets replaced by the associated value according to the local setting. The preceding code changes the locale several times using `setlocale()` and then calls `strftime()`. Note the differences that can be seen in Figure 3.1. Also take a look at Figure 3.2, in which the same script was executed on a Windows machine. According to the documentation, most of `strftime()` also works on Windows, but on some configurations changing the locale just does not seem to work. Therefore, it is very important to test first whether the system supports localized dates.
Automatically Localizing Dates

![Automatically Localizing Dates Image]

In (American) English: Wed 06 Apr 2005 01:54:06 PM CEST
In (British) English: Wed 06 Apr 2005 01:54:06 PM CEST
Auf Deutsch: Mit 06 Apr 2005 13:54:06 CEST
En Français: mer 06 apr 2005 13:54:06 CEST

Figure 3.1 The current date in different locales.

![Manually Localizing Dates Image]

In (American) English: 04/11/05 13:54:06
In (British) English: 04/11/05 13:54:06
Auf Deutsch: 04/11/05 13:54:06
En Français: 04/11/05 13:54:06

Figure 3.2 This particular system does not seem to support locales.

Manually Localizing Dates

```php
<?php
$weekdays = array(
    'domingo', 'lunes', 'martes', 'miércoles',
    'jueves', 'viernes', 'sábado'
);
$months = array(
    'enero', 'febrero', 'marzo', 'abril',
```
Localizing Dates Manually (localdate.php)

If you cannot rely on setlocale(), yet want to use localized date and time values, you have to do the translations by yourself and store the results in an array. Then, you can use date() to retrieve information about a date. This serves as an index for your array.

The preceding code does this for both the day of the month and the month itself. One array contains the Spanish weekdays; another one contains the month names.

Note that the value for the month is decreased by 1 because the array $months has no dummy element at position 0; therefore, month number one (January) has the index 0.

Using the Current Date the U.S./U.K./European Way

<?php
    echo 'mayo', 'junio', 'julio', 'agosto',
    'septiembre', 'octubre', 'noviembre',
    'diciembre'
);?
    $weekday = date('w');
    $month = date('n');
    echo $weekdays[$weekday] . date('j') .
    $months[$month - 1] . date('Y');
?>

Using the Current Date the U.S./U.K./European Way

<?php
    echo 'US format: ' . date('m/d/Y') .
    'UK format: ' . date('d/m/Y') .
    'German format: ' . date('d.m.Y') .
    'International format: ' . date('Y-d-m');
?>
Using the Current Date the U.S./U.K./European Way

Some International Date Formats (dateformats.php)
To give you a short and convenient reference, the preceding code contains several commonly used date formats. Depending on where you are, the order in which day, month, and year are used might vary:

- In the United States, it’s (mostly) month, day, and year
- In the United Kingdom and the rest of Europe, it’s (mostly) day, month, and year
- The international standard date notation starts with the year and continues with month and day

NOTE
The preceding code used a four-digit representation of the year because this is nonambiguous. In practice, however, two-digit years are also commonly used.

Formatting a Specific Date

```php
echo 'Time stamp: ' . mktime(12, 0, 0, 1, 1, 2001);
```

Creating a Time Stamp to be Used with Other Date/Time Functions (mktime.php)
All previous phrases in this chapter have used the current date and time. However, it is also possible to use any other arbitrary date and time value. For this, the relevant functions (especially `date()` and `strftime()`) accept a second parameter—a time stamp of the date to be analyzed. This time stamp is an integer value, the date and time in the so-called epoch format. This is the number of seconds that passed since January 1st, 1970, midnight GMT—the beginning of the UNIX
epoche. The time stamp/epoche value of the current moment can be retrieved using time() or by calling date('U') (see Table 3.1).

If you want to use another date, mktime() comes into play. This function converts a date provided by year, month, day, hour, minute, and second into an epoche value. Probably the strangest thing about this function is the order in which these parameters are passed:

1. Hour
2. Minute
3. Second
4. Month
5. Day
6. Year

You can also provide an optional seventh parameter, regardless of whether it is Daylight Savings Time (DST). This is relevant close to the change from or to DST because that does not happen at the same time over the world.

The preceding code calculates the time stamp for noon on the first day of this millennium. Because there was no year 0, this was January 1st, 2001.

NOTE
If you love collecting useless facts: On September 9, 2001, at precisely 3:46:40 a.m., the time stamp was 1,000,000,000. Therefore, mktime(3, 46, 40, 9, 9, 2001) returns 1000000000 (if your time zone is Central Europe).
Validating a Date

Validating a Date

```
<?php
if (checkdate(2, 29, 2000)) {
    echo '2000 was a leap year.';
} else {
    echo '2000 was not a leap year.';
}
if (checkdate(2, 29, 2100)) {
    echo '2100 will be a leap year.';
} else {
    echo '2100 will not be a leap year.';
}
?>
```

Validating Date Information (checkdate.php)

When you get a date—for example, from the user using an HTML form—this data must be validated. This includes checking whether the month exists and if the month has enough days. If it's February, you might also want to find out whether it is a leap year.

But PHP would not be PHP if you really had to do this on your own. The function checkdate() validates a date; you provide the month, the day, and the year as parameters.

Calculating a Relative Date

```
$expiry = time() + 30 * 24 * 60 * 60; //30 days
echo strftime('%c', $expiry);
```
Sometimes, you have the task of calculating a date that is relative to the current date, for example, “30 days from now.” Of course, you could put some real work into actually calculating this value, taking into account which day it is, whether it’s a leap year, and whether DST is relevant.

Far easier is the use of an epoch time stamp. Take, for instance, the aforementioned task of finding a date that lies 30 days in the future. One day has 24 hours, one hour has 60 minutes, and one minute has 60 seconds. Therefore, to get the current time stamp (using `time()` or `date('U')`), you just need to add 30 \* 24 \* 60 \* 60, and you have the time stamp of the desired date. This time stamp can then be used to set a cookie’s expiry date or just to print out some information about this date.

**TIP**

Why was 2000 a leap year, but 2100 not? The definition says: If a year is divisible by 4 and is either not divisible by 100 or is divisible by 400, it is a leap year and February has 29 days (this is because the Earth needs approximately 365.25 days to revolve around the sun). If you want to determine whether a given year is a leap year, this function comes in handy:

```php
function isLeapYear($year) {
    return ($year % 4 == 0 &&
            ($year % 100 != 0 || $year % 400 == 0));
}
```
Creating a Sortable Time Stamp

Converting a Date into a (Sortable) Time Stamp (timestamp.php)

Sometimes, using date values with a database is not a clever thing. Different language versions of the database, its drivers, or the underlying operating system could cause some trouble when the regional date formats do not fit together.

A potential alternative is the use of time stamps. Several different formats are available, but most of them have the following structure: year-month-day-hours-minutes-seconds. Using this value order, the string representation of the date can be easily sorted, which allows using it in databases.

To create such a time stamp, just a special call to date() is required. In the preceding code, this is encapsulated in a function for easy reuse.

```php
function timestamp($t = null) {
    if ($t == null) {
        $t = time();
    }
    return date('YmdHis', $t);  
}

<?php
    function timestamp($t = null) {            
        if ($t == null) {
            $t = time();
        }
        return date('YmdHis', $t);
    }

    echo 'Time stamp: ' . timestamp(time());
?>
```
Converting a String into a Date

TIP
This format is also used by MySQL for the representation of its TIMESTAMP data type.

Converting a String into a Date

'Yesterday: ' . date('r', strtotime('Yesterday'))

<?php
echo 'Yesterday: ' . date('r', strtotime('Yesterday')) . '<br />
echo 'Today: ' . date('r', strtotime('Today')) . '<br />
echo 'Tomorrow: ' . date('r', strtotime('Tomorrow')) . '<br />
echo 'In one week: ' . date('r', strtotime('+1 week')) . '<br />
echo 'One month before: ' . date('r', strtotime('-1 month')) . '<br />
echo 'Last Sunday: ' . date('r', strtotime('Last Sunday')) . '<br />
echo 'Next fantasy day: ' .
   var_export(@date('r', strtotime('Next fantasy day')), true);
?>

Converting a String into a Date (strtotime.php)

Previously, you saw a numeric representation of a date—either a triplet of day, month, and year, or a timestamp value. This time, you can go the other way and convert a string representation of a date/time value into an epoch value or something else that is usable within PHP and its date/time functions.
Converting a String into a Date

The whole magic is put into the PHP function 
strtotime(). According to the documentation, it “parse[s] about any English textual date/time description into a UNIX time stamp.” It sounds amazing, and it is amazing. The basis for this is the GNU date syntax; the code at the beginning of this phrase shows some examples for strtotime().

NOTE
At the time of this writing, strtotime() shows some strange behavior when a relative date is calculated and a change from or to DST is imminent. Also at the time of this writing, PHP’s date/time functions are about be rewritten and amended.

Determining Sunrise and Sunset

date_sunrise(time(), SUNFUNCS_RET_STRING, 48, 11.5, 90, 1)
date_sunset(time(), SUNFUNCS_RET_STRING, 48, 11.5, 90, 1);

```php
<?php
    echo 'Sunrise: ' .
        date_sunrise(time(), SUNFUNCS_RET_STRING, 48, 11.5, 90, 1) . '<br>';
    echo 'Sunset: ' .
        date_sunset(time(), SUNFUNCS_RET_STRING, 48, 11.5, 90, 1);
?>
```
Calculating when Sun Rises and Sets (sun.php)

Depending on the current location and date, the times for sunrise and sunset can drastically vary. However, formulas exist for determining this value depending on latitude and longitude, and PHP has this functionality integrated into its core starting with PHP 5. All you are required to do is call `date.sunrise()` and `date.sunset()`. Both functions expect a number of parameters:

- A time stamp (epoch value) of the date for which to determine the sunrise/sunset
- The desired format for the return value: `SUFUNCS_RET_DOUBLE` returns the time as a float value (between 0 and 23.99), `SUFUNCS_RET_STRING` returns it as a string (between 00:00 and 23:59), and `SUFUNCS_RET_TIMESTAMP` returns an epoch value
- The latitude (Northern latitude; use negative values for a Southern latitude)
- The longitude (Eastern longitude; use negative values for a Western longitude)
- The zenith of the sunrise (in degrees)
- The offset (in hours) to Greenwich mean time (GMT)

So, the preceding code calculates the sunrise and sunset for Munich, Germany, which resides at about 48° Northern latitude, 11° 30' Eastern longitude, for the current day. I checked it: It worked!
Using Date and Time for Benchmarks

Using Date and Time for Benchmarks

```php
<?php
// ...

$start = microtime(true);
$s = '';
for ($i=0; $i < 100000; $i++) {
    $s .= "$i";
}
$end = microtime(true);
echo 'Using double quotes: ' . ($end-$start) . '<br />

$start = microtime(true);
$s = '';
for ($i=0; $i < 100000; $i++) {
    $s .= $i;
}
$end = microtime(true);
echo 'Using no quotes: ' . ($end-$start) . '<br />

?>
```

Benchmarking Code Using microtime()
(benchmark.php; excerpt)

Up to now, all date/time functions did not produce results that were more precise than on the second level; no microseconds were available. This changes when you use the function gettimeofday(). This returns an array of values. The key 'sec' returns the
Using Date and Time for Benchmarks

associated epoche value; however, 'usec' returns the additional microseconds. With this, a very exact value can be used for operations that need exact measurement, for example, benchmarks.

The code at the beginning of this phrase contains a function microtimestamp() that returns an exact time stamp. This function is called twice; in between, a more or less complex calculation is done. The result of this is a benchmark that might help decide which coding technique is superior.

```php
function microtimestamp() {
    $timeofday = gettimeofday();
    return $timeofday['sec'] + $timeofday['usec'] / 1000000;
}
```

Figure 3.3 shows the result. Your mileage might vary, but you will find that using the double quotes just costs time. (In another experiment, you might also find out that it makes no difference whether you use single or double quotes—despite popular beliefs.)

![Figure 3.3](image)

**Figure 3.3** The second piece of code is the faster one.
Using Form Fields for Date Selection

```php
<?php
for ($i = 1; $i <= 12; $i++) {
    $monthname = date('F', mktime(12, 0, 0, $i, 1, 2005));
    echo "<option value="$i">$monthname</option>";
}
?>
<form method="post" action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
    <select name="day">
    for ($i = 1; $i <= 31; $i++) {
        if ($i <= 9) {
            echo "<option value="$i">0$ı</option>";
        } else {
            echo "<option value="$i">$i</option>";
        }
    }
    ?></select>
    <select name="month">
    for ($i = 1; $i <= 12; $i++) {
        $monthname = date('F', mktime(12, 0, 0, $i, 1, 2005));
        echo "<option value="$i">$monthname</option>";
    }
    ?></select>
    <select name="year">
    for ($i = 2005; $i <= 2010; $i++) {
        echo "<option value="$i">$i</option>";
    }
    ?></select>
    </form>
```
If you want to offer an HTML form to select a date, like many hotel- and flight-booking services offer, you can use the various parameters of date(); loop through all months of the year; and, therefore, create a selection list of days, months, and years. The preceding code contains the code for this; see Figure 3.4 for the result.

Create Self-updating Form Fields for Date Selection

```php
<?php
    if (isset($_POST['month']) && is_numeric($_POST['month']) && ((int)$_POST['month'] >= 1 && (int)$_POST['month'] <= 12)) {
        $month = (int)$_POST['month'];
    } else {
        $month = date('n');
    }
```
Create Self-updating Form Fields for Date Selection

```php
if (isset($_POST['year']) && is_numeric($_POST['year']) &&
    ((int)$_POST['year'] >= 2005 &&
    (int)$_POST['year'] <= 2010)) {
    $year = (int)$_POST['year'];
} else {
    $year = date('Y');
}

<?php
<form method="post" action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
    <select name="day">
        <?php
            $maxdays = date('t', mktime(12, 0, 0, $month, 1, $year));
            for ($i = 1; $i <= $maxdays; $i++) {
                if (isset($_POST['day']) && $_POST['day'] == $i) {
                    $sel = ' selected';
                } elseif ($i == date('j')) {
                    $sel = ' selected';
                } else {
                    $sel = '';
                }
                echo '<option value="$i"'.$sel.'>$i</option>

            }
        ?></select>

    // ...
</form>
```

The Date Selection Updates Itself Automatically
(dateSelection-js.php; excerpt)

The code from the previous phrase has one minor flaw: The number of days per month is always from 1 to 31, even in months that have less days. Using JavaScript, it is possible to write a fancy script that calculates how many days the current month has and then updates the selection list.
However, it is much more convenient to use a combination of JavaScript and PHP. Using JavaScript, you automatically submit the form. Using PHP, you prefill the form fields (see Chapter 4, “Interacting with Web Forms,” for more information about that) and, more importantly, find out how many days the related year has.

The JavaScript code is limited to a minimum: Selecting another month submits the HTML form (as does selecting another year because leap years make February longer):

```html
<select name="month" onchange='this.form.submit();'>
...<select name="year" onchange='this.form.submit();'>
```
The number of days per month can be found using `date('t')`. The listing at the beginning of this phrase contains the complete code for this, including some sanity checks for the information transmitted. Also, the code automatically preselects the current date, unless the user chooses something else. Figure 3.5 contains the output of the complete code.

**Calculating the Difference Between Two Dates**

```php
<?php
$century = mktime(12, 0, 0, 1, 1, 2001);
$today = time();
$difference = $today - $century;

echo 'This century started ';
echo floor($difference / 84600);
$difference -= 84600 * floor($difference / 84600);

echo 'days, ';
echo floor($difference / 3600);
$difference -= 3600 * floor($difference / 3600);

echo 'hours, ';
echo floor($difference / 60);
$difference -= 60 * floor($difference / 60);

echo 'minutes, and ' . $difference . ' seconds ago.';
?>
```

The Difference Between Two Dates (timediff.php)
Calculating the Difference Between Two Dates

The PHP code filled the month selection list with the appropriate number of days.

The epoch value that can be determined by `time()` and other PHP functions can be used to easily calculate the difference between two dates. The trick is to convert the dates into timestamps (if not already available in this format). Then the difference between these two timestamps is calculated. The result is the time difference in seconds. This value can then be used to find out how many minutes, hours, and days this corresponds to:

- Divide the result by 60 to get the number of minutes
- Divide the result by 60 * 60 = 3600 to get the number of hours
- Divide the result by 60 * 60 * 24 = 86400 to get the number of days
Calculating the Difference Between Two Dates

If you start with the number of days, round down each result and subtract this from the result; you can also split up the difference into days, hours, and minutes.

Using GMT Date/Time Information

Usually, PHP takes the local settings for time formats. However in some special cases, the GMT time format must be used. For this, PHP offers “GMT-enabled” versions of some of its date/time functions:

- `gmdate()` works like `date()` and formats a date/time value; however, the return value uses GMT format
- `gmmktime()` creates a time stamp like `mktime()`; however, it uses GMT
- `gmstrftime()` formats a time as `strftime()` does; however, it uses GMT

GMT is important when it comes to setting a page’s expiry date in an HTTP header or manually setting the expiry date of a cookie, also in the HTTP header.

What Does PEAR Offer?
The following PEAR packages offer functionality helpful for processing form data of any kind:

- Date contains a set of functions to work with various date/time values, including conversions between time zones and various date/time representations
- Date_Holidays calculates the names and dates of special holidays