15.3 Datum und Uhrzeit formatieren

15.3.1 Das Problem

Mit Datum und Uhrzeit zu arbeiten ist eines der komplizierteren Themen in der Programmierung. Lachen Sie nicht! Es hängt wesentlich mehr an dem Thema, als es den Anschein hat. Ein Datum einfach als einen String zu sehen, wie man zuerst vermuten würde, reicht bei Weitem nicht aus.

Denn wenn Sie ein Datum als eine Aneinanderreihung von Zeichen sehen, ohne dass PHP den Sinn dahinter kennt, verlieren Sie viele Möglichkeiten. Nehmen wir zum Beispiel den String `'25.07.2003 15:37'`, der ein typisches Datum mit Uhrzeit darstellt.

Sie können nur mit viel Mühe einen Teil des Datums ausgeben, zum Beispiel die Uhrzeit. Sie müssen den String mit `explode()` oder mit `substr()` (siehe Abschnitt 15.5.2) zerlegen, was nicht nur aufwendig, sondern auch fehleranfällig ist. Was ist beispielsweise, wenn ein anderer String zuerst die Uhrzeit und dann das Datum enthält?

Ein weiteres Problem sind die Zusatzinformationen, die für jedes Datum existieren, aber nicht direkt im String zu lesen sind. So war der 25.07.2003 zum Beispiel ein Freitag. Versuchen Sie das mal mit eigenem PHP-Code aus dem oben gezeigten Datumsstring herauszulesen. Viel Vergnügen!

Wie genau lösen Sie das Problem, wenn ich Ihnen folgende PHP-Aufgabe stelle: Zu dem Beispiel-Datum sollen 7 Tage addiert werden. Auf den ersten Blick ist das relativ einfach: Die Zahl 25, also den Tag isolieren (z. B. mit `explode()`) und dann 7 addieren. Schon haben wir den `32.07.2003`. Denken Sie mal darüber nach.

Von Themen wie der Lokalisierung von Datumsausgaben wollen wir erst gar nicht reden. In England wird ein Datum traditionell `07-25-2003`, also Monat-Tag-Jahr, geschrieben. Das ISO-standardisierte Datum ist Jahr-Monat-Tag, also `2003-07-25`. Den obigen Datumsstring so zu zerlegen und wieder korrekt zusammenzubauen ist zwar möglich, aber ziemlich aufwendig.

Die eben genannten Beispiele haben Ihnen hoffentlich gezeigt, dass dieses Thema doch ein wenig mehr Aufmerksamkeit benötigt.

15.3.2 Der Timestamp

Wir benötigen also ein Datumsformat, mit dem wir rechnen, das wir beliebig formatieren und für verschiedene Zeitzonen ausgeben können. Mit Dingen wie Wochentagen, Monatslänge und Schaltjahren sollte es am besten auch noch umgehen können.

Zu diesem Zweck wurde schon vor vielen Jahren ein Format festgelegt, das alle diese Anforderungen erfüllt. Das beste ist: Es ist erstaunlich einfach. Der sogenannte ***Unix Timestamp***, oder kurz ***Timestamp***, ist die Anzahl der Sekunden seit Donnerstag, dem 01.01.1970 0:00 Uhr UTC.[17] Dieses Jahr und der Name wurde ausgewählt, weil etwa zu diesem Zeitpunkt das Betriebssystem *Unix* erfunden wurde, welches dieses Zeitformat zum ersten Mal verwendet hat. Dieses Startdatum wird deshalb auch als ***The Epoch*** oder ***Unix-Epoche*** bezeichnet.

17. An vielen Stellen findet man anstatt UTC (Coordinated Universal Time) noch die Abkürzung GMT (Greenwich Mean Time), womit normalerweise die Zeitzone UTC + 0 (Westeuropäische Zeit, kurz WEZ) gemeint ist.

Wir haben es also mit einer simplen Integerzahl zu tun, was uns viele Vorteile bringt. Wenn Sie zum Beispiel zu einem Zeitpunkt eine Stunde addieren wollen, müssen Sie einfach 3600 (60 * 60, da eine Stunde 60 Minuten und jede Minute wiederum 60 Sekunden hat) addieren. Um herauszufinden, welche von zwei Datumsangaben die spätere ist, müssen Sie einen einfachen Zahlenvergleich durchführen, also beispielsweise `$timestamp1 > $timestamp2`.

15.3.3 time()

Um den aktuellen Timestamp zu erhalten, gibt es in PHP die Funktion `time()`. Wie festgelegt, gibt diese die Anzahl der Sekunden seit Beginn der Unix-Zeitrechnung (das heißt wirklich so) als Integerzahl zurück.

```
int time ( void )
```

Beispiel

```
1 <?php
2
3 echo time();
```

Codebeispiel 190 *time.php*

Die Ausgabe des Skripts war bei meinem ersten Test des Listings `1211535397`. Versuchen Sie ruhig einmal herauszufinden, um welches Datum es sich dabei handelt.

15.3.4 strftime()

Sie kennen nun zwar die Vorteile des Timestamp-Formats, der Nachteil ist aber ebenfalls sehr deutlich. Niemand kann auf Anhieb sagen, welches Datum sich hinter einem Timestamp verbirgt. Wir benötigen also eine Funktion, die aus einem Timestamp etwas Lesbares zaubert.

```
string strftime ( string $format [, int $ Timestamp ] )
```

Die Funktion `strftime()` erfüllt diese Aufgabe. Sie können dieser Funktion als ersten Parameter einen Formatstring (ähnlich `vprintf()`) übergeben, der festlegt, welche Elemente des Datums und der Uhrzeit ausgegeben werden sollen. Der zweite Parameter ist der Timestamp, für den ein Datum formatiert werden soll. Wenn Sie den zweiten Parameter weglassen, wird der aktuelle Timestamp verwendet. Sie können also Folgendes schreiben, um das aktuelle Datum zu formatieren:

```
strftime('format-das-ich-gleich-erklaere');
```

statt des längeren

```
strftime('format-das-ich-gleich-erklaere', time());
```

Der Formatstring

Der Formatstring arbeitet ähnlich, wie Sie es bereits von `vprintf()` kennen. Sie haben im Prinzip einen normalen String, der einige Platzhalter enthält, die in der Ausgabe durch ihre Werte ersetzt werden. Auch hier beginnen diese wieder mit einem Prozentzeichen.

Eine kleine Auswahl der existierenden Platzhalter ist:

- %d: Tag des Monats (1 - 31)
- %m: Monat (1 - 12)
- %Y: Vierstellige Jahreszahl (z. B. 2005)
- %H: Zweistellige Stunden im 24-Stunden-Format (z. B. 18)
- %M: Zweistellige Minuten
- %S: Zweistellige Sekunden

Wenn Sie also die aktuelle Uhrzeit (ohne Sekunden) ausgeben wollen, würden Sie Folgendes schreiben:

```
echo strftime('%H:%M');
```

Das würde zum Beispiel `13:35` ausgeben.

Beispiel

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title>Zeitansage</title>
</head>

<body>
    <h1>Zeitansage</h1>
    <p>
        Heute ist der <?php echo strftime('%d.%m.%Y'); ?>.
    </p>
    <p>
        Beim nächsten Ton ist es
        <?php echo strftime('%H Uhr %M und %S Sekunden'); ?>.
    </p>
    <p>
        <strong>BEEEP</strong>
    </p>
</body>

</html>
```

Codebeispiel 191 *strftime.php*

Beachten Sie, wie in den beiden `strftime()`-Aufrufen normale Zeichen und sogar Wörter mit den Platzhaltern vermischt werden, um eine gut lesbare Ausgabe zu erzeugen.

```
Warning: strftime(): It is not safe to rely on the system's timezone
settings (...)
```

Spätestens seit Version PHP 5.3 ist es wichtig geworden, dass PHP vor der Nutzung solcher Funktionen eine korrekte Zeitzonenangabe erhält. Dies können Sie beispielsweise für ein einzelnes Skript mit dem Funktionsaufruf `date_default_timezone_set('Europe/Berlin')` erreichen. Der String `'Europe/Berlin'` ist natürlich nur eine Möglichkeit von vielen. Eine Liste der unterstützten Zeitzonen finden Sie unter http://php.net/manual/de/timezones.php.

Zeitzone in der php.ini anpassen

Besser ist es jedoch, wenn Sie den korrekten Wert global in der *php.ini* Ihrer Entwicklungsumgebung bzw. des Webservers hinterlegen. Bei manchen Hosting-Anbietern ist eine Änderung der *php.ini* leider nicht möglich. Nutzen Sie in diesem Fall den Funktionsaufruf. Ob die Konfigurationsanpassung auf einem Webserver bereits erfolgt ist, erfahren Sie mit der Funktion `phpinfo()` im Abschnitt **date** beim Punkt **Default timezone**.

Ist dort nicht die gewünschte Zeitzone angegeben, so müssen Sie in der aktiven *php.ini* nach der Angabe `date.timezone` suchen. Diese ist seit PHP 5.4 normalerweise standardmäßig vorhanden. Sollte dies nicht der Fall sein, müssen Sie sie selbst ergänzen.

```
date.timezone=Europe/Berlin
```

Geben Sie nach dem Gleichheitszeichen die gewünschte Zeitzone an und achten Sie darauf, dass die Zeile nicht mit einem Semikolon beginnt. Auch diese Änderung wird wie schon die Anpassung der Fehlerausgabe in Abschnitt 5.3.3 nicht sofort aktiv. Zuerst muss Ihr Webserver neu gestartet werden.

15.3.5 mktime()

Der Timestamp ist zwar ein perfektes, universelles Austauschformat für Datumsangaben, nur können Sie schlecht Ihre Benutzer dazu auffordern, Datumswerte in diesem Format einzugeben. Was denken Sie, wie viele Leute sich auf Ihren Seiten registrieren würden, wenn Sie dort lesen müssten:

```
Geben Sie in dieses Feld bitte Ihr Geburtsdatum als Unix-Timestamp ein. Der
Timestamp ist die Anzahl der Sekunden seit Beginn der Unix-Zeitrechnung.
Wenn Sie vor diesem Zeitpunkt geboren sind, müssen Sie einen negativen Wert
eingeben.
```

Das wird so nicht funktionieren. Seien Sie froh, wenn Ihre Benutzer überhaupt halbwegs vernünftige Daten eingeben.

Wir benötigen also eine Möglichkeit, einen Datumsstring wie `24.12.2005` in einen Timestamp umzuwandeln. Verwenden wir hierfür testweise die Ihnen bereits bekannte Funktion `explode()` und die neue Funktion `mktime()`, welche Datumswerte in einen Timestamp umwandeln kann.

```
int mktime ([ int $hour [, int $minute [, int $second [, int $month [, int
$day [, int $year ]]]]] ] )
```

Jeder Parameter steht bei `mktime()` für einen anderen Teil einer Datumsangabe, wobei theoretisch alle optional sind.[18] Wird ein Parameter nicht angegeben, so wird als Standardwert der aktuelle Zeitpunkt für diesen Parameter zugrunde gelegt. Wenn Sie nur Parameter setzen wollen, die weiter hinten stehen, so können Sie die vorderen mit `0` oder `null` belegen.

Lassen Sie mich einige Beispiele zeigen, um die Funktion etwas zu verdeutlichen:

Um den Jahreswechsel `2008-2009`, also `01.01.2009 00:00:00` als Timestamp zu erhalten, müssen Sie

```
mktime(0, 0, 0, 1, 1, 2009);
```

eingeben.

Den Timestamp für eine Minute vor diesem Jahreswechsel müssen Sie als

```
mktime(23, 59, 0, 12, 31, 2008);
```

angeben. Achten Sie darauf, dass in der Funktion der Monat vor dem Tag steht.

Datumsstrings zerlegen

Nun können Sie schlecht 6 Textfelder in Ihr HTML-Formular einbauen, um alle diese Werte einzeln zu erhalten. Für gewöhnlich legt man maximal zwei Textfelder an, einen für das Datum und einen für die Uhrzeit (sofern letztere benötigt wird).

Also müssen Sie die beiden Strings noch weiter zerlegen, um an die einzelnen Werte zu gelangen. Das Problem, dass Benutzer unter Umständen auch falsch formatierte oder unvollständige Werte eingeben, wollen wir mit Rücksicht auf Ihre geistige Gesundheit ignorieren.

Wir gehen also von zwei Benutzereingaben aus, die folgendermaßen aufgebaut sind:

```
$datum = '17.04.2006';
```

```
$uhrzeit = '23:05:00';
```

Stellen Sie sich vor, diese würden aus einer Benutzereingabe stammen.

Zuerst müssen Sie die beiden Strings mit `explode()` in ihre Bestandteile zerlegen. Als Trenner verwenden Sie bei `$datum` den Punkt und bei `$uhrzeit` den Doppelpunkt.

Dann müssen Sie der Funktion `mktime()` die Parameter in der richtigen Reihenfolge übergeben und den Timestamp auffangen.

18. Praktisch führt eine Nutzung ganz ohne Parameterbefüllung im Gegensatz zu `time()` zu einer Warnmeldung von PHP.

Beispiel

```
1  <?php
2
3  $datum = '17.04.2006';
4  $uhrzeit = '23:05:00';
5
6  $datumTeile = explode('.', $datum);
7  $uhrzeitTeile = explode(':', $uhrzeit);
8
9  $timestamp = mktime(
10     $uhrzeitTeile[0],
11     $uhrzeitTeile[1],
12     $uhrzeitTeile[2],
13     $datumTeile[1],
14     $datumTeile[0],
15     $datumTeile[2]
16 );
17
18 echo strftime('%Y-%m-%d %H:%M', $timestamp);
```

Codebeispiel 192 *mktime1.php*

Da die einzelnen Parameter von `mktime()` nicht auf eine Zeile gepasst hätten, habe ich jeden Parameter in eine einzelne Zeile geschrieben, was in PHP erlaubt ist. Beachten Sie jedoch, dass Sie im Gegensatz zu Arrays kein Komma hinter dem letzten Parameter notieren dürfen (Zeile 15).

Eine kleine Verbesserung können wir noch vornehmen, wenn wir `list()` [19] zu Hilfe nehmen. Hiermit kann ein Array in einzelne Variablen zerlegt werden, was häufig in Verbindung mit Funktionen benötigt wird, die ein Array als Rückgabewert haben.

Statt die Werte in einer einzigen Variablen aufzufangen, können Sie mit `list()` eine Variable für jedes Array-Element, das geliefert wird, angeben.

Statt

```
$datumTeile = explode('.', $datum);
```

schreiben Sie nun

```
list($tag, $monat, $jahr) = explode('.', $datum);
```

wobei in `$tag` das Array-Element mit Index `0` landet, in `$monat` das mit Index `1` und in `$jahr` das mit Index `2`. Nun müssen Sie bei `mktime()` nicht mehr mit den einzelnen Array-Indizes hantieren, sondern können aussagekräftige Variablennamen verwenden. Das sieht nicht nur besser aus, es besteht auch weniger die Gefahr, Parameter in der falschen Reihenfolge zu verwenden, was gerade bei Monat und Tag sehr gefährlich ist.

Das überarbeitete Beispiel sieht so aus:

19. Wie `array()` ist auch dies keine wirkliche Funktion, sondern ein Sprachkonstrukt.

Beispiel

```php
<?php

$datum = '17.04.2006';
$uhrzeit = '23:05:00';

list($tag, $monat, $jahr) = explode('.', $datum);
list($stunde, $minute, $sekunde) = explode(':', $uhrzeit);

$timestamp = mktime(
    $stunde,
    $minute,
    $sekunde,
    $monat,
    $tag,
    $jahr
);

echo strftime('%Y-%m-%d %H:%M', $timestamp);
```

Codebeispiel 193 *mktime2.php*

Zugegeben, es ist kein riesiger Unterschied, aber die Lösung ist besser lesbar.

Da Sie einen solchen Umwandlungscode von Strings in Timestamps bestimmt häufiger benötigen, bietet er sich geradezu für eine Funktion an.

Beispiel

```php
<?php

function ermittleTimestamp($datum, $uhrzeit)
{
    list($tag, $monat, $jahr) = explode('.', $datum);
    list($stunde, $minute, $sekunde) = explode(':', $uhrzeit);

    $timestamp = mktime(
        $stunde,
        $minute,
        $sekunde,
        $monat,
        $tag,
        $jahr
    );

    return $timestamp;
}

$datum = '17.04.2006';
$uhrzeit = '23:05:00';

echo strftime(
    '%Y-%m-%d %H:%M',
    ermittleTimestamp($datum, $uhrzeit)
);
```

Codebeispiel 194 *strtotime1.php*

15.3.6 strtotime()

Vielleicht ärgern Sie sich jetzt, wenn ich Ihnen erzähle, dass eine solche Umwandlung auch mit viel weniger Code geht. Doch dies wird zukünftig oftmals ein Problem für Sie sein, da es viele sehr spezialisierte Funktionen in PHP gibt, die Sie (noch) nicht kennen. Sie beginnen also oftmals, das Rad neu zu erfinden. Erst sehr viel später merken Sie oder einer Ihrer Kollegen vielleicht, dass er hierfür bereits eine bessere Lösung gibt. Wenn Ihr Code gut strukturiert ist, lässt sich dieses Problem jedoch normalerweise mit wenigen Änderungen am Code lösen. In diesem Fall müssen wir beispielsweise lediglich den Inhalt der Funktion `ermittleTimestamp()` anpassen.

```
int strtotime ( string $time [, int $now ] )
```

Beispiel

```php
<?php

function ermittleTimestamp($datum, $uhrzeit)
{
    return strtotime($datum . ' ' . $uhrzeit);
}

$datum = '17.04.2006';
$uhrzeit = '23:05:00';

echo strftime(
    '%Y-%m-%d %H:%M',
    ermittleTimestamp($datum, $uhrzeit)
);
```

Codebeispiel 195 *strtotime2.php*

Da wir immer noch von getrennten Datums- und Zeitwerten ausgehen und die Funktion `strtotime()` nur einen einzigen String im ersten Parameter erwartet, müssen wir beide Strings zunächst verketten.

Eine noch etwas bessere und zuverlässigere Alternative zu `strtotime()` ist übrigens die Nutzung der `DateTime`-Klasse (siehe http://php.net/manual/de/datetime.construct.php). Diese sollten Sie sich allerdings erst anschauen, wenn Sie sich schon eine Weile mit objektorientierter Programmierung beschäftigt haben.