Rechnen mit Timestamps, Daten und Uhrzeiten in PHP

Im Folgenden werde ich auf das Rechnen mit Timestamps am Beispiel einer Funktion besprechen, die einen Timestamp schneller als date() bzw. getdate() in Datum, Uhrzeit, Wochentag etc. umrechnet. Dabei werde ich einige Schwierigkeiten ansprechen, die einem das Rechnen mit Timestamps schwierig machen können.

Vorraussetzung zum Verständnis dieses Textes sind grundlegende Kenntnisse der Sprache PHP sowie Allgemeinwissen.
Sie werden viel über Zeit lernen, z.B. dass nicht jeder Tag 24 Stunden hat und nicht jedes Jahr 365 Tage.

Was ist ein Timestamp?

Ein „Timestamp“ ist ein Zeitstempel nach „Unixzeit“. Er gibt die Anzahl Sekunden ab Beginn der Unix-Epoche, also dem 01.01.1970 um 00:00:00 (Weltzeit) an. Die erste Schwierigkeit die sich hierbei auftut, zeigt sich, wenn man folgenden PHP-Code ausführt:

```
echo date('d.m.Y – H:i:s',0);
```

Die Funktion date() wandelt einen Timestamp (2. Parameter, hier 0) in ein menschenlesbares Format um (Form wird über 1. Parameter festgelegt). Hier soll Datum und Uhrzeit des Timestamps 0 ausgegeben werden. Man erwartet also folgende Ausgabe:

```
01.01.1970 – 00:00:00
```

Die tatsächliche Ausgabe lautet aber:

```
01.01.1970 – 01:00:00
```

Dies ist zumindest die Ausgabe auf einem korrekt eingestellten Server in Deutschland. Hier wird also die Zeitzone korrekt beachtet – da wir der Weltzeit (UTC) eine Stunde voraus sind, entspricht der Timestamp 0 ein Uhr nachts am 01.01.1970 nach deutscher Zeit. Um Funktionen für den deutschen Raum zu schreiben sollte man dies also berücksichtigen.

Performance-Probleme mit date() und getdate()

Die Funktionen date() und getdate() sind ja eigentlich genau das, was Ziel dieses Artikels sein soll: Funktionen, die einen Timestamp in Datum und Uhrzeit, Wochentag, etc. umrechnen. Der Unterschied bei getdate ist, dass die Rückgabe in Form eines Arrays erfolgt und das Format nicht komfortabel mit einem String übergeben werden kann. Beim Umstellen des Logformats von CrazyStat sollte nun für jeden auszuwertenden Logeintrag der Timestamp in Datum, Wochentag und Uhrzeit umgewandelt werden um Speicherplatz zu sparen (Datum, Uhrzeit etc. wird nicht mehr abgespeichert). Dies führte dazu, dass CrazyStat mehr als die Hälfte der Auswertezeit mit dem Umrechnen von Timestamps verbrachte. Zuerst versuchte ich statt des mehrmaligen Aufrufs von date() auf den gleichen Timestamp getdate() zu verwenden. Ergebnis: Keinerlei Unterschied. Es scheint so, als würde date() intern die Funktion getdate() aufrufen, das Ergebnis entsprechend des String-Musters parsen und rückliefern. Getdate() wiederum scheint das Ergebnis zu Cachen, das wiederholte Aufrufen von getdate() auf den gleichen Zeitraum dauert nur unwesentlich länger als das einmalige Aufrufen. Dies bedeutet: Wenn man date(„Y“,0) aufruft, wird nicht nur das Jahr, sondern auch der Monat, der Tag, der Wochentag, die Sekunde usw. berechnet, obwohl dies gar nicht gefragt war. Ruft man danach date(„s“,0); auf, geht dies blitzschnell, da das Ergebnis bereits berechnet und gecacht vorliegt. Dies bedeutet aber auch, dass das zeitaufwändigere Berechnen der Sekunde erfolgt, obwohl man nur das Jahr berechnet bräuchte.

Daraus ergibt sich, dass die Funtionen date() und getdate() in den meisten Fällen unnötig Zeit brauchen, was bei tausendfachem Aufrufen ins Gewicht fällt. Selbst wenn man alle berechneten Werte benötigt, lässt sich, wie im folgendem gezeigt wird, das Ergebnis (in den meisten Fällen) schneller berechnen

Performance durch Lernen

Nach einigen Versuchen mit langsameren und ungenauen Ansätzen entschied ich mich für einen „lernenden“ Algorithmus.
Die Funktion, die erstellt werden soll, wird tausende Timestamps aus dem gleichen Zeitbereich, also innerhalb weniger Jahre, umwandeln müssen. Deshalb werden wir ihr ein gewisses „Grundwissen“ fest einprogrammieren, was Informationen über diesen Zeitraum enthält. Sollte der gewählte Zeitraum nicht ausreichen, so baut die Funktion ihr „Wissen“ selbstständig weiter aus.
Der Ansatz ist, dass wir der Funktion die Timestamps für den Monatsbeginn, also den jeweils ersten des Monats um 00:00:00 Uhr, für den Zeitraum von November 2005 (CrazyStat ist seit Dezember 2005 public) bis Januar 2010 als Grundwissen einspeichern:

```
// Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600,1133391600,1136070000,
1138748400,1141167600,1143842400,1146434400,1149112800,1151704
800,1154383200,1157061600,1159653600,1162335600,1164927600,116
7606000,1170284400,1172703600,1175378400,1177970400,1180648800
,1183240800,1185919200,1188597600,1191189600,1193871600,119646
3600,1199142000,1201820400,1204326000,1207000800,1209592800,12
12271200,1214863200,1217541600,1220220000,1222812000,122549400
0,1228086000,1230764400,1233442800,1235862000,1238536800,12411
28800,1243807200,1246399200,1249077600,1251756000,1254348000,1
257030000,1259622000,1262300400);
```

Das Array wird „static“ deklariert, damit die Funktion sich selbstberechnete Monatsbeginne merken und beim nächsten Aufruf verwenden kann (s. PHP-Doku).

Eine erste Umsetzung

Zuerst gehen wir davon aus, dass der Timestamp nach dem 01.11.2005 liegt (später wird diese Einschränkung aufgehoben.

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600,1133391600,1136070000,
1138748400,1141167600,1143842400,1146434400,1149112800,1151704
800,1154383200,1157061600,1159653600,1162335600,1164927600,116
7606000,1170284400,1172703600,1175378400,1177970400,1180648800
,1183240800,1185919200,1188597600,1191189600,1193871600,119646
3600,1199142000,1201820400,1204326000,1207000800,1209592800,12
12271200,1214863200,1217541600,1220220000,1222812000,122549400
0,1228086000,1230764400,1233442800,1235862000,1238536800,12411
28800,1243807200,1246399200,1249077600,1251756000,1254348000,1
257030000,1259622000,1262300400);

 $month=11;
 $year=2005;
 // get month and year
 for($y=0;$stamp>=$secs[$y];$y++)
```

```
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 }
```

Es wird eine performante for-Schleife verwendet, die jeden Monat durchgeht bis der Timestamp in diesem Monat liegt. Bei jedem Durchgang wird der Monat erhöht, sollte er danach 13 betragen, wird der Monat auf 1 gesetzt und das Jahr erhöht. Da aber die Schleife aber eigentlich einmal zu viel durchläuft, muss dies durch herabsetzen des Monats und evtl. des Jahrs korrigiert werden.

Das eigentliche Lernen

Die Funktion kann jetzt ausschließlich Timestamps im festgelegten Zeitraum umrechnen. Sollte der Timestamp nach 2010 liegen, erkennt diese Funktion das, errechnet den benötigten Wert und speichert ihn in $secs. Das Berechnen mittels mktime ist zwar zeitaufwendig, durch das Lernen fällt dies aber bei vielen Aufrufen nicht ins Gewicht.

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);

 $month=11;
 $year=2005;
 // get month and year
 for($y=0;$stamp>=$secs[$y];$y++)
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  if(!isset($secs[$y+1]))
   $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 }
```

Beachten Sie, dass mktime() hier einen Monat>=12 übergeben bekommt. Dies ist kein Problem, für mktime ist der 13. Monat in 2005 gleich dem 1. Monat in 2006. Somit wird nur eine Variable übergeben, was wohl performanter ist (nicht anders getestet).
Hiermit ist unsere Funktion für alle Timestamps ab 11/2005 geeignet. Im folgendem soll sie für alle gültigen Timestamps erweitert werden.

Berechnen von Jahr und Monat für beliebige Timestamps

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);

 // before 1.1.96 (slow and bad) => use getdate
 if($stamp<820450800)
  {
  $result=getdate($stamp);
  return array($result['year'],$result['mon']);
  }
// if timestamp>=(timestamp of 1.1.2010), start 2010
 elseif($stamp>=1262300400)
  {
  $month=1;
  $year=2010;
  $y=50;
  }
// if timestamp>=(timestamp of 1.1.2007), start 2007
 elseif($stamp>=1167606000)
  {
  $month=1;
  $year=2007;
  $y=14;
  }
 // normally start 11/2005
 elseif($stamp>=1130799600)
  {
  $month=11;
  $year=2005;
  $y=0;
  }
 // between 1/2000 and 11/2005 => start 1/2000
 elseif($stamp>=946681200)
  {
  $month=1;
  $year=2000;
  $y=-70;
  $secs[$y]=946681200;
  }
 // between 1/1996 and 1/2000 => start 1/1996
 else
  {
  $month=1;
  $year=1996;
```

```
  $y=-118;
  $secs[$y]=820450800;
  }
 // get month and year
 for(;$stamp>=$secs[$y];$y++)
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  if(!isset($secs[$y+1]))
   $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 return array($year,$month);
 }
```

Der Timestamp wird jetzt zuerst in einen Bereich eingeordnet. Timestamps vor dem 01.01.1996 werden nicht selbst umgerechnet sondern per getdate, da Tests gezeigt haben, dass eine solche Erweiterung nur mit erweitertem „Grundwissen“ sinnvoll wäre. Beachten Sie, dass return die Funktion abbricht. Es gibt noch eine weitere Schwierigkeit, die vor 1996 auftritt und später behandelt wird.
Die Reihe if-Möglichkeiten könnte theoretisch unendlich ausgedehnt werden, allerdings dauert das Ausrechnen nicht so lange, dass dies eine Selektion mit sehr vielen Möglichkeiten rechtfertigt. Hier muss ein gesunder Mittelwert gefunden werden.
Interessant ist hier, wie die for-Schleife ohne den ersten Ausdruck verwendet wird (dieser wird ja in der Selektion gesetzt).
Auch zu beachten ist, dass numerische Schlüssel aus Performance-Gründen für das Rückgabe-Array gewählt wurden (anders als bei getdate). Dies lässt sich natürlich nach Bedarf anpassen.
Diese Funktion ist zum Errmitteln von Monat und Jahr fehlerfrei und schneller als getdate(), allerdings wäre für diese simple Aufgabe eventuell ein rein mathematischer, nicht lernender Algorithmus schneller. Im Folgenden soll diese Funktion auch die Uhrzeit, den Tag des Monats und den Wochentag ermitteln.

Tag des Monats

Die folgende Funktion ermittelt zusätzlich den Tag des Monats:

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);

 // before 1.1.96 (slow and bad) => use getdate
 if($stamp<820450800)
```

```
  {
  $result=getdate($stamp);
  return array($result['year'],$result['mon'],
$result['mday']);
  }
// if timestamp>=(timestamp of 1.1.2010), start 2010
 elseif($stamp>=1262300400)
  {
  $month=1;
  $year=2010;
  $y=50;
  }
// if timestamp>=(timestamp of 1.1.2007), start 2007
 elseif($stamp>=1167606000)
  {
  $month=1;
  $year=2007;
  $y=14;
  }
 // normally start 11/2005
 elseif($stamp>=1130799600)
  {
  $month=11;
  $year=2005;
  $y=0;
  }
 // between 1/2000 and 11/2005 => start 1/2000
 elseif($stamp>=946681200)
  {
  $month=1;
  $year=2000;
  $y=-70;
  $secs[$y]=946681200;
  }
 // between 1/1996 and 1/2000 => start 1/1996
 else
  {
  $month=1;
  $year=1996;
  $y=-118;
  $secs[$y]=820450800;
  }
 // get month and year
 for(;$stamp>=$secs[$y];$y++)
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  if(!isset($secs[$y+1]))
```

```
   $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 $y--;
 // seconds of the month
 $monthsecs=$secs[$y];
 // seconds from beginning of the month until stamp
 $totalsecs=$stamp-$monthsecs;

 // Calculate Day
 $day=ceil($totalsecs/86400);
 if($day<1) $day=1;
 return array($year,$month,$day);
 }
```

Der Ansatz ist simpel: Die Sekunden, die seit Monatsbeginn vergangen sind werden durch die Anzahl Sekunden pro Tag (24h*60min*60s=86400s) geteilt, dass Ergebnis aufgerundet. Falls $stamp genau die erste Sekunde eines Monats ist, sind in diesem Monat erst 0 Sekunden vergangen, dadurch würde der Tag 0 herauskommen, dies wird mit der Selektion korrigiert.

Wenn man diese Funktion mit einem automatisierten Test überprüft, wird bei 100.000 zufällig gewählten Timestamps der Tag in 0.07 % der Fälle falsch ermittelt (Vergleich mit getdate). Wo der Fehler liegt? Auffällig ist, dass die Fehler immer Ende März und Ende Oktober zwischen 23:00 und 01:00 Uhr auftreten. Klingelt's?
Nein? Sommerzeit! Diese wird bei den Monat-Anfangs-Timestamps in $secs bereits berücksichtigt, allerdings kommt es zu Fehlern, wenn ein Timestamp nach der Umstellung auf Sommerzeit (immer Ende März) aber noch im März liegt. Gleiches gilt im Oktober. Denn nicht jeder Tag hat 24 Stunden! Jedes Jahr hat einer 23 und einer 25!

Sommerzeit

Wir verfahren hier wie vorher und setzen einen lernenden Algorithmus ein. Den Beginn und das Ende der Sommerzeit in 2006-2009 speichern wir als „Grundwissen", fehlende Werte wird die Funktion (zeitintensiv) errechnen und sich merken.
Wann beginnt denn eigentlich die Sommerzeit, und wann endet sie? Wie bereits erwähnt, beginnt sie im März, genauer am letzten Sonntag, und endet am letzten Sonntag im Oktober. Dies ist zumindest in Mitteleuropa der Fall, für Amerika, Russland und viele andere Staaten gelten andere aber ähnliche Regeln, manche Staaten haben gar keine Sommerzeit.
Die Sommerzeit macht auch das Problem für Timestamps vor 1996 aus: Denn vor 1996 dauerte die Sommerzeit nur bis Ende September.

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);
 // Daylight Saving Time start and end in 2006-2009
```

```
 static $dst_start=array(2006=>1143334800,2007=>1174784400,
2008=>1206838800,2009=>1238288400);
 static $dst_end=array(2006=>1162083600,2007=>1193533200,
2008=>1224982800,2009=>1256432400);

 // before 1.1.96 (slow and bad) => use getdate
 if($stamp<820450800)
  {
  $result=getdate($stamp);
  return array($result['year'],$result['mon'],
$result['mday']);
  }
// if timestamp>=(timestamp of 1.1.2010), start 2010
 elseif($stamp>=1262300400)
  {
  $month=1;
  $year=2010;
  $y=50;
  }
// if timestamp>=(timestamp of 1.1.2007), start 2007
 elseif($stamp>=1167606000)
  {
  $month=1;
  $year=2007;
  $y=14;
  }
 // normally start 11/2005
 elseif($stamp>=1130799600)
  {
  $month=11;
  $year=2005;
  $y=0;
  }
 // between 1/2000 and 11/2005 => start 1/2000
 elseif($stamp>=946681200)
  {
  $month=1;
  $year=2000;
  $y=-70;
  $secs[$y]=946681200;
  }
 // between 1/1996 and 1/2000 => start 1/1996
 else
  {
  $month=1;
  $year=1996;
  $y=-118;
  $secs[$y]=820450800;
  }
 // get month and year
 for(;$stamp>=$secs[$y];$y++)
  {
```

```
   $month++;
   if($month==13)
    {
    $month=1;
    $year++;
    }
   if(!isset($secs[$y+1]))
    $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
   }
  $month--;
  if($month==0)
   {
   $month=12;
   $year--;
   }
  $y--;
  // Daylight Saving Time
  if($month==3 && !isset($dst_start[$year]))
   $dst_start[$year]=mktime(2,0,0,3,31-date('w',
mktime(2,0,0,3,31,$year)),$year);
  elseif($month==10 && !isset($dst_end[$year]))
   $dst_end[$year]=mktime(2,0,0,10,31-date('w',
mktime(2,0,0,10,31,$year)),$year);

  if($month==3 && $stamp>=$dst_start[$year])
   $monthsecs=$secs[$y]-3600;
  elseif($month==10 && $stamp>=$dst_end[$year])
   $monthsecs=$secs[$y]+3600;
  else $monthsecs=$secs[$y]; // seconds of the month

  // seconds from beginning of the month until stamp
  $totalsecs=$stamp-$monthsecs;

  // Calculate Day
  $day=ceil($totalsecs/86400);
  if($day<1) $day=1;
  return array($year,$month,$day);
  }
```

Zuerst werden die Timestamps, an denen die Sommerzeit beginnt und endet der Jahre 2006 bis 2009 als „Grundwissen“ abgelegt (wieder static). Die eigentliche Korrektur erfolgt über $monthsecs. Wenn $stamp im März oder Oktober nach der Zeitumstellung liegt, wird die Anzahl Sekunden des Monats um eine Stunde angepasst.
Außerdem ist direkt schon vorgesehen, dass das „Wissen“ durch Ausrechnen weiterer Sommerzeit-Beginn- und End-Stamps erweitert wird. Der Timestamp des Sommerzeit-Beginns wird wie folgt ermittelt:
date('w', mktime(2,0,0,3,31,$year) ermittelt den Wochentag des 31. März als Zahl (0=Sonntag, 6=Samstag). Dann wird mit **mktime(2,0,0,3,31-**date('w', mktime(2,0,0,3,31,$year))**,$year);** der Timestamp um 2:00 Uhr am Schalttag - nämlich dem 31. Minus dem Wochentag des 31. - errechnet. Wenn der 31. ein Sonntag ist, wird also 0 abgezogen, der 31 ist Schaltag (da letzter Sonntag im März). Ist der 31. ein Samstag, wird 6

abgezogen, der 25. ist Schalttag (letzter Sonntag). Die Umstellung auf Sommerzeit erfolgt stets um 02:00 Uhr.
Die Ermittlung des Sommerzeit-Endes erfolgt analog für den Oktober.

Ermitteln der Uhrzeit

Als nächstes soll das Errechnen der Uhrzeit eingefügt werden:

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);
 // Daylight Saving Time start and end in 2006-2009
 static $dst_start=array(2006=>1143334800,2007=>1174784400,
2008=>1206838800,2009=>1238288400);
 static $dst_end=array(2006=>1162083600,2007=>1193533200,
2008=>1224982800,2009=>1256432400);

 // before 1.1.96 (slow and bad) => use getdate
 if($stamp<820450800)
  {
  $result=getdate($stamp);
  return array($result['year'],$result['mon'],
$result['mday']);
  }
// if timestamp>=(timestamp of 1.1.2010), start 2010
 elseif($stamp>=1262300400)
  {
  $month=1;
  $year=2010;
  $y=50;
  }
// if timestamp>=(timestamp of 1.1.2007), start 2007
 elseif($stamp>=1167606000)
  {
  $month=1;
  $year=2007;
  $y=14;
  }
 // normally start 11/2005
 elseif($stamp>=1130799600)
  {
  $month=11;
  $year=2005;
  $y=0;
  }
 // between 1/2000 and 11/2005 => start 1/2000
 elseif($stamp>=946681200)
  {
  $month=1;
  $year=2000;
```

```
  $y=-70;
  $secs[$y]=946681200;
  }
 // between 1/1996 and 1/2000 => start 1/1996
 else
  {
  $month=1;
  $year=1996;
  $y=-118;
  $secs[$y]=820450800;
  }
 // get month and year
 for(;$stamp>=$secs[$y];$y++)
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  if(!isset($secs[$y+1]))
   $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 $y--;
 // Daylight Saving Time
 if($month==3 && !isset($dst_start[$year]))
  $dst_start[$year]=mktime(2,0,0,3,31-date('w',
mktime(2,0,0,3,31,$year)),$year);
 elseif($month==10 && !isset($dst_end[$year]))
  $dst_end[$year]=mktime(2,0,0,10,31-date('w',
mktime(2,0,0,10,31,$year)),$year);

 if($month==3 && $stamp>=$dst_start[$year])
  $monthsecs=$secs[$y]-3600;
 elseif($month==10 && $stamp>=$dst_end[$year])
  $monthsecs=$secs[$y]+3600;
 else $monthsecs=$secs[$y]; // seconds of the month

 // seconds from beginning of the month until stamp
 $totalsecs=$stamp-$monthsecs;
 // seconds from the beginning of the day until stamp
 $daysecs=$totalsecs%86400;

 // Calculate Day
 $day=ceil($totalsecs/86400);
 if($day<1) $day=1;
```

```
 // calculate hour
 $hour=floor($daysecs/3600);
 // calculate minute
 $minute=floor(($daysecs%3600)/60);
 // calculate second
 $second=($daysecs%3600)%60;

 return array($year,$month,$day,$hour,$minute,$second);
 }
```

Zuerst werden die Sekunden seit Mitternacht berechnet ($daysecs). Dazu dient der Rest der Ganzzahldivision (Modulo) der Sekunden seit Monatsbeginn ($totalsecs) durch die Anzahl Sekunden eines Tages.
Die Stunde wird dann berechnet, indem die Sekunden seit Tagesbeginn durch die Sekunden einer Stunde (60*60=3600) geteilt und das Ergebnis abgerundet wird.
Für die Minute wird mittels Modulo die Sekunden seit Stundenbeginn ausgerechnet und dies dann durch 60 geteilt (abgerundet). Die Sekunde ergibt sich aus der gleichen Rechnung, der Rest dieser Division sind nämlich genau die Sekunden.

Wochentag

Der Wochentag wird wieder über ein lernendes Verfahren ermittelt. Dazu werden die Wochentage des Monatsbeginns abgelegt. Um den Wochentag zu bestimmen addiert man den Tag des Monats minus 1 mit dem Wochentag des Monatsanfangs. Wenn man diesen Wert durch 7 (Anz. Tage/Woche) teilt, ergibt der Rest den Wochentag. Ist der Wochentag des Monatsgbeginns nicht bekannt, wird er über date() gelernt.
Das Verfahren ist ähnlich wie bei den anderen Werten.

```
function getDateFast($stamp)
 {
 // Seconds of the months 11/2005 to 1/2010 (1st 00:00:00)
static $secs=array(1130799600, …, 262300400);
 // weekdays of the timestamps above
 $wdays=array(2        ,4        ,0        ,3        ,3
,6        ,1        ,4        ,6        ,2        ,5
,0        ,3        ,5        ,1        ,4        ,4
,0        ,2        ,5        ,0        ,3        ,6
,1        ,4        ,6        ,2        ,5        ,6
,2        ,4        ,0        ,2        ,5        ,1
,3        ,6        ,1        ,4        ,0        ,0
,3        ,5        ,1        ,3        ,6        ,2
,4        ,0        ,2        ,5        );

 // Daylight Saving Time start and end in 2006-2009
 static $dst_start=array(2006=>1143334800,2007=>1174784400,
2008=>1206838800,2009=>1238288400);
 static $dst_end=array(2006=>1162083600,2007=>1193533200,
2008=>1224982800,2009=>1256432400);

 // before 1.1.96 (slow and bad) => use getdate
 if($stamp<820450800)
```

```
  {
  $result=getdate($stamp);
  return array($result['year'],$result['mon'],
$result['mday']);
  }
// if timestamp>=(timestamp of 1.1.2010), start 2010
 elseif($stamp>=1262300400)
  {
  $month=1;
  $year=2010;
  $y=50;
  }
// if timestamp>=(timestamp of 1.1.2007), start 2007
 elseif($stamp>=1167606000)
  {
  $month=1;
  $year=2007;
  $y=14;
  }
 // normally start 11/2005
 elseif($stamp>=1130799600)
  {
  $month=11;
  $year=2005;
  $y=0;
  }
 // between 1/2000 and 11/2005 => start 1/2000
 elseif($stamp>=946681200)
  {
  $month=1;
  $year=2000;
  $y=-70;
  $secs[$y]=946681200;
  }
 // between 1/1996 and 1/2000 => start 1/1996
 else
  {
  $month=1;
  $year=1996;
  $y=-118;
  $secs[$y]=820450800;
  }
 // get month and year
 for(;$stamp>=$secs[$y];$y++)
  {
  $month++;
  if($month==13)
   {
   $month=1;
   $year++;
   }
  if(!isset($secs[$y+1]))
```

```
   $secs[$y+1]=mktime(0,0,0,$y+12,1,2005);
  }
 $month--;
 if($month==0)
  {
  $month=12;
  $year--;
  }
 $y--;
 // Daylight Saving Time
 if($month==3 && !isset($dst_start[$year]))
  $dst_start[$year]=mktime(2,0,0,3,31-date('w',
mktime(2,0,0,3,31,$year)),$year);
 elseif($month==10 && !isset($dst_end[$year]))
  $dst_end[$year]=mktime(2,0,0,10,31-date('w',
mktime(2,0,0,10,31,$year)),$year);

 if($month==3 && $stamp>=$dst_start[$year])
  $monthsecs=$secs[$y]-3600;
 elseif($month==10 && $stamp>=$dst_end[$year])
  $monthsecs=$secs[$y]+3600;
 else $monthsecs=$secs[$y]; // seconds of the month

 // seconds from beginning of the month until stamp
 $totalsecs=$stamp-$monthsecs;
 // seconds from the beginning of the day until stamp
 $daysecs=$totalsecs%86400;

 // Calculate Day
 $day=ceil($totalsecs/86400);
 if($day<1) $day=1;

 // calculate hour
 $hour=floor($daysecs/3600);
 // calculate minute
 $minute=floor(($daysecs%3600)/60);
 // calculate second
 $second=($daysecs%3600)%60;
 // Calculate weekday
 if(!isset($wdays[$y])) $wdays[$y]=date('w',$secs[$y]);
 $weekday=($day+$wdays[$y]-1)%7;
 return
array($year,$month,$day,$hour,$minute,$second,$weekday);
 }
```

Abschließender Kommentar

Eine .php-Datei zum Download, welche die getDateFast-Funktion enthält, finden Sie hier: http://www.christosoft.de/index.php?download=getDateFast.zip
Das Erweitern der Funktion auf Timestamps vor 1996 wäre der nächste logische Schritt zur Vervollständigung der Funktion. Dazu lasse man die Sommerzeit in diesen Fällen einfach am letzten Sonntag im September um 3.00 Uhr enden.