
# PHP für Einsteiger

PHP UserGroup Hannover, 10. Oktober 2002

Von Ulrich Hacke <hacke@trilos.de>

Trilos IT-Dienstleistungen GbR
Am Rathaus 15
30592 Ronnenberg
fon 0511 – 21 44 98 – 60
fax 0511 – 21 44 98 – 65
http://www.trilos.de
service@trilos.de

# Inhaltsverzeichnis

# 1 Einleitung

PHP steht für "Hypertext Preprocessor" und ist eine Scriptsprache zur Erstellung dynamischer Webseiten. Die PHP-Sprachkonstrukte sind in die HTML-Struktur eines Webdokumentes eingebunden; ihre Syntax ist der von Java, C und Perl ähnlich. PHP läuft auf allen gängigen UNIX- und Windows-Versionen. Als CGI arbeitet PHP mit jedem Webserver zusammen - für einige Server (z.B. den Apache) gibt es aber auch wesentlich effizientere Modulversionen. Heute ist PHP eine der beliebtesten und erfolgreichsten Programmiersprachen im weltweiten Netz.

Es wird davon ausgegangen, dass der Leser bereits über eine lauffähige Installation von PHP verfügt und sich mit den grundlegenden Notationsregeln auskennt. Besprochen werden in diesem Tutorial die Grundlagen, die für die Erstellung eigener Scripte nötig sind. Zusätzlich sind lesenswerte Texte über die Entstehung von PHP sowie über die Geschichte des Internets enthalten. Es sei darauf hingewiesen, dass dieses Tutorial, welches im Rahmen der PHP-Usergroup Hannover entstand, selbstverständlich keinen vollständigen PHP-Kurs ersetzen kann. Übrigens kann das komplette Tutorial auch unter http://www.hacke.net/php heruntergeladen werden, um in Ruhe offline betrachtet zu werden.

# 2 Sprachkonstrukte

Jede Programmiersprache besteht aus Sprachkonstrukten über welche die Abläufe innerhalb des Programms oder des Scripts gesteuert werden. In PHP verdeutlichen die erstellten Konstrukte dem Parser, was er zu tun hat.

Jedes PHP-Skript besteht aus einer Reihe von Anweisungen. Diese Anweisungen können Zuweisungen, Funktionsaufrufe, Schleifen, oder Kontrollstrukturen sein. Jeder Befehl ist in der Regel mit einem Semikolon abgeschlossen. Wie auch in anderen Sprachen - wie z.B. C, Perl oder Java - können Befehle zu einer Gruppe zusammengefasst werden, die durch geschweifte Klammern umschlossen ist.

## 2.1 if...else...elseif

Bedingungen, die den Programmablauf direkt beeinflussen, sind wohl das wichtigste und älteste Element einer jeden Programmiersprache. Ohne dieses Sprachkonstrukt könnte keinerlei vernünftige Software erstellt werden. Die einfachste IF-Bedingung sieht in PHP so aus:

```
if($variable) {
  ...code...
}
```

Eine vollständigere Konstruktion sähe so aus:

```
if($essen == "Eisbein" or $essen == "Schnitzel") {
  echo "typisch deutsch...";
} elseif ($essen == "Hamburger") {
  echo "amerikanisches fast food";
} else {
  echo "irgendwas anderes";
}
```

An Vergleichsoperatoren für zwei Variablen oder Therme $a und $b kennt PHP:

```
>               $a größer $b
<               $a kleiner $b
==              $a gleich $b
===             $a gleich $b (und beide vom gleichen Typ)
and bzw. &&     ergibt TRUE wenn $a UND $b wahr sind
or bzw. ||      ergibt TRUE wenn $a ODER $b wahr sind
!               negiert den Ausdruck
```

Hinweis: Ein beliebter Fehler ist es, einen vergleichenden Ausdruck als $a = $b zu schreiben (ein "=" vergessen). Egal was die Variablen für Werte haben - dieser Ausdruck ist IMMER wahr... :-)

Oft steht man vor einer Aufgabenstellung wie: wenn $a > 10 dann $b gleich "viel" sonst b gleich "wenig". Dafür bietet sich die Kurzschreibweise für Bedingungen an:

```
$b = ($a>10) ? "viel" : "wenig";
```

Das funktioniert auch als Ausgabe von echo oder als Rückgabewert einer Funktion.

## 2.2 for

Mit for wird eine Zählschleife erzeugt, die so oft durchlaufen wird, wie es ihr Zähle angibt. Die folgende Schleife wird zehn Mal durchlaufen:

```
for($i=0;$i<10;$i++) {
  echo $i ". Durchlauf<br>";
}
```

Eine for-Schleife kann jederzeit durch break verlassen werden. Man kann die Parameter übrigens auch leer lassen und sich im Schleifenblock um das Hochzählen oder das Abbruchkriterium kümmern.

Eine weitere interessante Konstruktion ist die foreach-Schleife, mit welcher bequem ein Array durchlaufen werden kann:

```
$arr = array("rot","grün","blau");
foreach($arr as $farbe) {
  echo "Farbe: " . $farbe . "<br>";
}
```

## 2.3 while

while sind kopfgesteuerte Schleifen, d.h. sie werden so oft durchlaufen, bis die übergebene Bedingung TRUE ist:

```
while($i<10) {
  echo $i;
  $i++;
}
```

Merke: while-Schleifen können auch ÜBERHAUPT NICHT durchlaufen werden - nämlich genau dann, wenn die Bedingung schon von Anfang an FALSE ist.
Eine interessante Anwendung ergibt sich in Verbindung mit list und each, wobei ersteres Variablen zuweist, als ob sie ein Array wären und letzteres das aktuelle Paar (Schlüssel und Wert) eines Arrays zurückliefert und dabei den internen Array-Pointer inkrementiert:

```
// Liefere alle Werte, die per POST übertragen wurden
while (list($schluessel, $wert) = each ($HTTP_POST_VARS)) {
  echo "$schluessel : $wert<br>";
}
```

Fast immer wird man while-Schleifen einsetzen, um das Ergebnis einer Datenbankabfrage zu verarbeiten (vergl. mySQL-Zugriff):

```
$query = mysql_query("select name from mitarbeiter", $dbh);
while($row = mysql_fetch_array($query)) {
  echo $row["name"] . "<br>";
}
```

while-Schleifen können jederzeit durch break verlassen werden.

## 2.4 do

do-Schleifen sind den while-Schleifen sehr ähnlich: hier wird die Abbruchbedingung allerdings erst im Fuß notiert, so dass do IMMER mindestens ein Mal durchlaufen werden:

```
$i = 1;
do {
  $i++;
while($i<10);
```

do-Schleifen können jederzeit durch break verlassen werden. Viele Programmierer machen davon Gebrauch, so zum Beispiel:

```
do {   // Ist $i groß genug?
  if($i<27) break;
  // Gültiger Bereich erreicht?   if($i>36) {
    ...$i bearbeiten...
    if($i>100) break;
  }
} while(0);
```

## 2.5 switch

Dieses Konstrukt wird immer dann angewendet, wenn viele Bedingungen ausgewertet werden müssen und die Handhabung von if-Sequenzen zu umständlich wäre. Man kann sich switch wie einen altertümlichen Drehschalter vorstellen:

```
switch($richtige_im_lotto) {
  case 3:
    echo "Naja, ganz brauchbar";
    Leckeren_Rotwein_kaufen();
    break;
  case 4:
    echo "Nicht schlecht, gar nicht schlecht!";
    Freundin_zum_Essen_einladen();
    break;
  case 5:
    echo "Freu! Tob!";
    Kurzurlaub_planen();
    break;
  case 6:
    echo "Waaaaahnsin! Brüll!";
    Ferrari_kaufen();
    Weltreise_machen();
    break;
  default:
    echo "Typisch...";
}
```

Der default-Block wird immer dann abgearbeitet, wenn keine der übrigen Bedingungen zutrifft. Besonders wichtig ist der break-Befehl in jedem case-Abschnitt: findet switch eine zutreffende Bedingung, wird - unmittelbar einsichtig - der zugehörige case-Block abgearbeitet. Alle folgenden case werden nicht mehr ausgewertet - der enthaltene Code wird aber ausgeführt!

## 2.6 include

include() dient zum Einbinden externer Dateien. Bei einem inlcude() wird vom PHP-Parsing-Modus in den HTML-Modus geschaltet, die angegebene Datei eingelesen und ausgewertet - danach wird wieder zurück in den Parser gesprungen. Je nach Konfiguration in der php.ini kann die angegebene Datei auch eine fremde URL sein. include() oder require() bieten sich an, um Codefragmente einzubinden und das eigentliche PHP-Skript kurz zu halten:

```
include("config.php");
include("functions.php");
include("classes.php");
```

**Worin besteht der Unterschied zwischen include() und require()?**

require() wird die angeforderte Datei IMMER einlesen, auch dann, wenn die Zeile gar nicht ausgeführt wird, in der das require() steht. Deshalb sollte man include() verwenden, wenn der Fall eintreten kann, dass die Datei gar nicht benötigt wird. Daraus resultiert auch, dass require() nicht in Schleifen eingesetzt werden darf.

Eine Datei, die mittels require() oder include() eingebunden wird, kennt die bis dato instanzierten Variablen und Objekte und kann damit arbeiten.

Um auszuschließen, dass eine Datei mehrmals eingebunden wird, kann man auch include_once() bzw. require_once() verwenden.

# 3 mySQL-Zugriff

mySQL ist sicherlich die am häufigsten in Verbindung mit PHP eingesetzte Datenbank. Man spricht auch von LAMP- (Linux, Apache, mySQL, PHP) oder WAMP-Systemen (Windows...).

PHP bringt eine ganze Reihe von mySQL-Funktionen mit, die eine schnelle und einfach Abfrage beliebiger Datenbanken ermöglichen. Um überhaupt mit einer mySQL-Datenbank arbeiten zu können, muss als erstes unter Zuhilfenahme von Datenbankbenutzer und -kennwort eine Verbindung zum mySQL-Server hergestellt werden. Die Verbindung wird über ein sogenanntes Datenbank-Handle gespeichert und identifiziert:

```
// Verbindung herstellen
$dbh = mysql_connect($DB_SERVER, $DB_USER, $DB_PASSWD);
mysql_select_db($DB_NAME);
```

Sollte die Verbindung fehlschlagen, wird mysql_connect() FALSE zurückliefern.

Alle Datenbank-Abfragen werden als SQL-Query an den Server gesendet:

```
$query = mysql_query($sql, $dbh);
```

SQL-Statements, die keinen Rückgabewert zurückliefern (wie z.B. INSERT oder CREATE) brauchen nicht mittels einer Zuweisung ausgewertet werden. Während auf Seiten von PHP eher wenig Kenntnisse benötigt werden, um einigermaßen effizient mit mySQL-Datenbanken arbeiten zu können, liegt die Hauptanforderung sicherlich in einem guten Datenbankdesign und weitreichenden SQL-Kenntnissen.

Beispiel: man habe eine Tabelle "mitarbeiter", welche die Felder "mitarbeiter_id", "name" und "vorname" enthalte. Gewünscht wird eine HTML-formatierte Liste alle Mitarbeiter. In PHP sähe das so aus:

```
$query = mysql_query("select mitarbeiter_id, name, vorname from mitarbeiter", $dbh);
if(mysql_affected_rows($dbh)>0) {
  echo "<table>";
  while($row = mysql_fetch_array($query)) {
     echo "<tr><td>" . $row["mitarbeiter_id"] . "</td><td>" . $row["name"] .
"</td><td>" . $row["vorname"] . "</td></tr>";
  }
} else {
  echo "Keine Mitarbeiter :-(";
}
```

mysql_fetch_array liefert einen Datensatz eines Ergebnis-Sets als assoziatives Array zurück, wobei die Schlüssel der Name des jeweiligen Datenbankfeldes darstellen. Alternativ kann auch mysql_fetch_row() verwendet werden - dann ist das Ergebnis ein numerisches Array.

In der Regel wird das SQL-Statement über verschiedene Bedingungen oder Anwender-Eingaben im PHP-Skript zusammengebaut. Je nach Aufbau der Datenbank kann der zu sendende SQL-String relativ komplex werden.

## 3.1 Weitere wichtige mySQL-Funktionen

**mysql_insert_id()**

- liefert die ID des neuen Datensatzes nach einer erfolgreichen INSERT-Anweisung, die für ein Feld vom Typ AUTO_INCREMENT vergeben wurde.

**mysql_result($query, $index, $feldname)**

- liefert den Wert des Feldes $feldname aus dem Ergebnis-Set $query an der Position $index. Hinweis: diese Funktion sollte nur dann verwendet werden, wenn das Ergebnis-Set sehr klein ist; die wesentlich leistungsfähigeren Funktion mysql_fetch_array() oder mysql_fetch_row() liefern einen ganzen Datensatz auf einmal und sind dadurch wesentlich schneller.

**mysql_error()**

- Liefert den Fehlertext des zuletzt ausgeführten mySQL-Statements und ist damit zum Debuggen ideal geeignet.

Es ist übrigens nicht notwendig, eine per mysql_connect() geöffnete Verbindung explizit über mysql_close() zu schließen, da bei Beendigung eines PHP-Skriptes alle offenen (nicht-persistenten) Verbindungen automatisch gekappt werden.

## 3.2 Tipps

**Auf Sicherheit setzen**

- Jeder Datenbankaufruf kann schief gehen. Fast alle mySQL-Funktionen können ein FALSE zurückliefern, wenn der gewünschte Effekt aus irgendeinem Grund nicht eingetreten ist. Über eine if-Konstruktion kann man das auswerten und ggf. in eine Fehlerbehandlungsroutine springen.

**Sonderzeichen entschärfen**

- Zeichen wie " (Anführungszeichen) oder ' (Hochkomma) können nicht einfach in ein SQL-Statement geschrieben werden, da sie u.U. das Ende des Strings kennzeichnen und die Datenbank durcheinander bringen. Alle diese Sonderzeichen müssen mit einem \ (Backslash) escaped werden. Dabei leisten die Funktionen addslashes() und stripslashes() gute Dienste.

  In der php.ini gibt es die Schalter MAGIC_QUOTES, die dieses Verhalten automatisieren können. Trotzdem sollte man sich angewöhnen, sich selbst um Sonderzeichen zu kümmern, damit die eigenen Scripte auf beliebigen PHP-Maschinen laufen, auf denen man zur php.ini keinen Zugang besitzt.

# 4 Funktionen

Neben den bereits vordefinierten Funktionen kann der Entwickler beliebige eigene Funktionen schreiben. Jede Funktion hat einen Namen, sowie (optional) eine Parameterliste und einen Rückgabewert:

```
function BierTrinken($anzahl_flaschen) {
  ...code...
  return ($status=="besoffen") ? true : false;
}
```

Es ist auch möglich, Vorgabewerte für Parameter zu vergeben, z.B. function ZeichneLinie($laenge, $farbe="rot) {}. Über die Funktionen func_num_args() und func_get_args() können Funktion sogar beliebig viele Parameter übergeben bekommen, ohne dies vor explizit definieren zu müssen.

Alle in einer Funktion verwendeten Variablen sind lokal, d.h. sie sind außerhalb der Funktion nicht bekannt. Das gilt auch für die Parameter: alle übergebenen Werte bleiben außerhalb unverändert - auch dann, wenn sie innerhalb der Funktion bearbeitet werden. Diese Art der Übergabe wird auch als "ByValue" bezeichnet. Soll ein Parameter jedoch tatsächlich als Verweis behandelt werden, muss ihm in der Parameterliste eine & (kaufmännisches "und") vorangestellt werden. Man spricht dann von einer Übergabe "ByReference".

Um innerhalb einer Funktion auf "äußere" Variablen zugreifen zu können, müssen sie über das Wort global erreichbar gemacht werden. Alternativ kann auch das spezielle PHP-Array $GLOBALS[] verwendet werden. Dieses assoziative Array ermöglicht den Zugriff auf alle globalen Variablen, indem man ihren Namen als Schlüssel verwendet.

Funktionen sollten häufig eingesetzt werden, frei nach dem Motto: "Alles was mehr als einmal aufgerufen wird, gehört in eine Funktion.". Dadurch lässt sich mittels Kapselung schlankerer Code schreiben. Oft benötigte Funktionen werden in einem separaten Script zusammengefasst und - falls benötigt - eingebunden.

Funktionen können sich natürlich auch selbst aufrufen, so dass rekursive Operationen möglich werden - wie beispielsweise die Darstellung eines binären Baumes oder dem Auslesen einer Verzeichnisstruktur.

## 4.1 CALLBACK-Funktionen

Obwohl Sie als "besonders" gelten, unterscheiden sich CALLBACK-Funktionen um keinen Deut von "normalen" PHP-Funktionen. Einige PHP-Sprachkonstrukte, wie z.B. array_filter() benötigt eine selbstgeschriebenen Funktion, um seine Aufgaben erfüllen zu können. Diese wird wie gehabt erstellt und per Name übergeben. Eine solche CALLBACK-Funktion muss auf jeden Fall einen Rückgabewert besitzen; in der Regel werden TRUE/FALSE-Konstruktionen eingesetzt.

# 5 Klassen

Obwohl sich PHP durch seine Fähigkeit, mit Klassen umgehen zu könne, von vielen anderen Programmiersprachen abhebt, werden Java- oder C++-Programmierer wahrscheinlich ein wenig enttäuscht sein. Es gibt keine strenge "Vererbungslehre" oder Polymorphismus wie in C++. Das macht auch insofern nicht so viel Sinn, wenn man sich vor Augen hält, dass alle definierten Klassen-Objekte nach der Abarbeitung eines PHP-Skriptes wieder zerstört werden und nur temporär im RAM des Servers enthalten sind. Dennoch sind Klassen mächtige Werkzeuge bei der Arbeit mit umfangreicheren Projekten.

Klassen sollte man sich als beliebige Objekte vorstellen, welche Eigenschaften und Methoden haben. Nehmen wir ein Auto als ein Objekt, dann hätte es beispielsweise die Eigenschaften "Baujahr", "Farbe", "Tankinhalt" usw., während Methoden Bremsen() oder Beschleunigen() sein könnten.

Die folgende Klasse definiert eine Person mit den Eigenschaften "Name" und "Geburtsdatum". Als Methode wird alter() implementiert, die das aktuelle Alter in Tagen zurückliefert.

```
// Eine Klasse fuer Personen
class person {

  // Eigenschaften
  var $name;
  var $geburtsdatum;

  // Konstruktor
  function person($name, $geburt) {
    $this->name = $name;
    $this->geburtsdatum = $geburt;
  }

  // Alter ermitteln
  function alter() {
    $result = (time() - $this->geburtsdatum) / 86400;
    return $result;
  }
}

// Arbeiten mit der Klasse
$p = new person("Ulrich", mktime(0,0,06,6,1972));

echo $p->name . " ist " . $p->alter . " Tage alt.";
```

Wenn man ein Objekt instanziert, möchte man in der Regel eine feste Ausgangssituation haben, d.h. es sollen bereits Eigenschaften bei der Erstellung übergeben werden können. Dazu bedient

man sich eines Konstruktors - in PHP ist das eine Klassenfunktion, die den gleichen Namen tragen muss wie die zugehörige Klasse. Ein wie in Java oder C++ benötigter Destruktor fällt hier weg, da - wie bereits erwähnt - alle Objekte nach Ablauf eines Skriptes sowieso zerstört werden.

Innerhalb einer Klasse wird auf die Eigenschaften und Methoden über das Schlüsselwort $this zugegriffen. Ein Pfeil (->) trennt das Schlüsselwort von der Methode oder Eigenschaft. Ein beliebter Fehler ist es übrigens, beim Abruf einer Eigenschaft die Eigenschaft selbst mit einem $ zu schreiben, da es sich ja eigentlich um eine Variable handelt. Anmerkung am Rande: selbstverständlich können Eigenschaften auch über variable Variablen abgerufen werden, z.B. $this->${"eigenschaft" . $i}.

Jede Klasse kann beliebig viele Eigenschaften und Methoden besitzen. Für Methoden gilt im großen und ganzen das Gleiche wie für Funktionen.

# 6 Dateioperationen

Mit den hier beschriebene Funktionen kann PHP auf das lokale Filesystem des Servers zugreifen. Manchmal bieten sich (Text)-Dateien als Alternative zur Datenbank an; entweder dann, wenn es nur geringe Datenmengen zu verwalten gibt oder der Einsatz einer Datenbank nicht möglich ist (nicht jeder Provider bietet das an).

Man unterscheidet (zumindest unter Windows) zwischen ASCII- und Binärdateien.

## 6.1 Lesen und Schreiben

Um auf eine Datei zugreifen zu können (egal ob sie gelegen oder erstellt werden soll), muss zunächst ein Zeiger (Filepointer) gesetzt werden:

```
// Datei öffnen
if($fp = fopen("datei.txt", "r")) {
  while(!feof($fp)) {
    $zeile = fgets($fp, 8192);
    ...$zeile verarbeiten
  }
} else {
  echo "Konnte Datei nicht öffnen";
}
```

Der zweite Parameter in fopen() gibt den Modus an, in welchem die Datei behandelt werden soll:

r öffnet die Datei nur zum Lesen und positioniert den Dateizeiger auf den Anfang der Datei

r+ Öffnet die Datei zum Lesen und Schreiben und setzt den Dateizeiger auf den Anfang der Datei.

w Öffnet die Datei nur zum Schreiben und setzt den Dateizeiger auf den Anfang der Datei sowie die Länge der Datei auf 0 Byte. Wenn die Datei nicht existiert wird versucht sie anzulegen.

w+ Öffnet die Datei zum Lesen und Schreiben und setzt den Dateizeiger auf den Anfang der Datei sowie die Länge der Datei auf 0 Byte. Wenn die Datei nicht existiert, wird versucht sie anzulegen.

a Öffnet die Datei nur zum Schreiben. Positioniert den Dateizeiger auf das Ende der Datei. Wenn die Datei nicht existiert, wird versucht sie anzulegen.

a+ Öffnet die Datei zum Lesen und Schreiben. Positioniert den Dateizeiger auf das Ende der Datei. Wenn die Datei nicht existiert, wird versucht sie anzulegen.

Um die Datei explizit als Binär-Datei zu behandeln, muss (unter Windows) der Buchstabe "b" im Modus hinzugefügt werden.

fgets($fp, $max) liest maximal $max Zeichen oder bricht früher ab, wenn es vorher auf einen Zeilenumbruch oder das Dateiende stößt. Letzteres kann per feof($fp) abgefragt werden und eignet sich hervorragend als Abbruchkriterium einer while-Schleife. Das Gegenstück zu fgets() ist fputs()

Eine Besonderheit bietet fopen(): wenn statt einer lokalen Datei eine URL übergeben wird, wird von dieser gelesen. Damit können beispielsweise Newsinhalte von fremden Sites ausgewertet werden.

## 6.2 Dateioperationen - Kopieren, Löschen, Umbenennen

**copy($quelle, $ziel)**

- Kopiert die angegebene Datei $quelle nach $ziel. Sollte der Kopiervorgang fehlschlagen, liefert copy() ein FALSE zurück. Die beiden Dateiparameter können auch Pfadangaben enthalten.

**rename($alt, $neu)**

- Benennt $alt in $neu um. Sollte dies fehlschlagen, wird ein FALSE zurückgegeben. $datei kann auch Pfadangaben enthalten.

**unlink($datei)**

- Versucht, $datei zu löschen und liefert FALSE, wenn das nicht geht. $datei kann auch Pfadangaben enthalten.

## 6.3 Dateioperationen - Verzeichnisse

Um Verzeichnisse bequem auslesen zu können, bringt PHP die Klasse dir() mit:

```
$d = dir("c:/MeineTexte");
while($inhalt = $d->read()) {
  echo $entry . "<br>";
}
$d->close();
```

Hinweis: mit dem obigen Code werden alle Inhalte des Verzeichnisses ausgelesen - das können neben Dateien natürlich auch weitere Verzeichnisse sein. Zusätzlich werden die "Spezialeinträge" "." (aktuelles Verzeichnis) und ".." (Verzeichnis der nächsthöheren Ebene) mit aufgefunden.

Über chdir() wird in ein anderes Verzeichnis gewechselt, mkdir() erstellt eine neues Verzeichnis ab der aktuellen Position im Dateibaum und rmdir() löscht eines.

Generell gilt: als Entwickler muss man auf die möglicherweise einschränkende Rechtestruktur auf dem Server achten. Unter UNIX kann man bei vielen Verzeichnisfunktionen auch den Mode (ugo-Konvention) als oktalen Wert mit übergeben. Unter Windows ist die Arbeit mit Verzeichnissen generell einfacher, da es in der Regel weniger Restriktionen gibt.

# 7 Beispiele

Die folgenden Beispiele sind nicht zwangsläufig von mir selbst programmiert, sondern sind frei im Internet verfügbar. Einige dienen nur der Demonstration von Programmiertechniken, während andere dem Entwickler das tägliche Leben einfacher machen.

## 7.1 HTML-konformen Text erzeugen

Bietet sich z.B. an, um beliebigen Text aus einer Datenbank an den Browser zu senden

```
function HTML($text, $flag=false) {
  $text = htmlentities(stripslashes($text));
  if($flag) $text = nl2br($text);
}
```

Schließlich sollte es selbstverständlich sein, alle "extended characters" als HTML-Äquivalent auszugeben. Unglaublich, wie viele Programmierer (sogar auf großen renommierten Sites) das vergessen. Zur Funktion: wenn $flag == true werden alle Zeilenumbrüche nach <br> konvertiert.

## 7.2 Datei aus HTTP-Upload entgegennehmen

Man habe ein Formular, über welches ein Anwender eine Datei (z.B. ein Bild) auf den Server laden kann. Der folgende Code nimmt den Datenstrom entgegen:

```
if(($datei!="")&&($datei!="none")) {
  $buffer = addslashes(fread(fopen($datei, "r"), filesize($datei)));
  // und nun beispielsweise in ein Datenbankfeld schreiben

  // Alternative: Datei umbenennen und wegkopieren
  $filename = md5($datei . time());
  copy($datei, $filename);
}

<form action="myscript.php3" method="post" enctype="multipart/form-data">
<input type="file" name="datei">
<input type="submit">
</form>
```

Um auch große Dateien via HTTP-Upload transferieren zu können, muss ggf. die php.ini angepasst werden. Wichtig sind hier die Einträge POST_MAX_SIZE, UPLOAD_MAX_FILESIZE und MAX_EXECUTIONTIME.

## 7.3 Datumsausgabe

Wenn man einen Datumswert aus einer mySQL-Datenbank ausliest, hat dieser das Format YYYY-MM-DD. Um das in einer europäischen Form auszugeben, bedient man sich der folgenden Funktion:

```
function WriteDate($zeile) {
  $mliste = array("Januar", "Februar", "März", "April", "Mai", "Juni", "Juli",
"August", "September", "Oktober", "November", "Dezember");
  return (substr($zeile, 8, 2) . ". " . $mliste[(substr($zeile, 5, 2) -1)] . " " .
substr($zeile, 0, 4));
}
```

Alternativ kann natürlich auch die passende SQL-Syntax von mySQL verwendet werden oder man arbeitet gleich mit Timestamps, die sogar den Vorteil haben, dass man Zeiträume zwischen zwei Datumsangaben sehr leicht berechnen kann. Aber das ist ein anderes (und vielschichtiges) Thema... :-)

## 7.4 Hyperlinks automatisch erzeugen

Die folgende Funktion macht aus URLs und eMail-Adressen in einer Zeichenkette klickbares HTML:

```
function click($text) {
  $result = eregi_replace(\"([[:alnum:]]+)://([^[:space:]]*)([[:alnum:]#?/&=])\",
\"\\\\1://\\\\2\\\\3\", $text); $ret = eregi_replace(\"(([a-z0-9_]|\\\\-
|\\\\.)+@([^[:space:]]*)([[:alnum:]-]))\", \"\\\\1\", $ret);
  return($ret);
}
```

Aus "Schicken Sie eine Mail an hacke@trilos.de oder statten Sie uns unter http://www.trilos.de einen Besuch ab!" wird "Schicken Sie eine Mail an hacke@trilos.de oder statten Sie uns unter http://www.trilos.de einen Besuch ab!"

## 7.5 eMail-Adresse validieren

Diese Funktion liefert TRUE, wenn $emailaddress eine syntaktisch korrekte Adresse ist.

```
function isMail($emailaddress) {
  return ((preg_match(\"/(@.*@)|(\\.\\.)|(@\\.)|(\\.@)|(^\\.)/\", $emailaddress) ||
preg_match(\"/^.+\\@(\\[?)[a-zA-Z0-9\\-\\.]+\\.([a-zA-Z]{2,3}|[0-
9]{1,3})(\\]?)$/\",$emailaddress)));
}
```

Das ist besonders dann praktisch, wenn man einem Anwender eine eMail an die Adresse schicken will, die er in ein Formularfeld eingeben soll (man glaubt es kaum, wie viele Adressen es nach dem Muster "krümelmonster@t-online" gibt )...

Noch schöner wäre es natürlich, wenn die Funktion auch gleich prüfen würde, ob die eMail-Adresse auch gültig ist... nun, wer macht's? ;-)

## 7.6 Nützliches zu Arrays

Schon fast trivial: wie befülle ich schnell und einfach ein Array?

```
// Alle Zeilen einer Textdatei in ein Array $lines einlesen.
$lines = array();
while(!feof($fp)) $lines[] = fgets($fp, 16384);
echo "Das waren " . count($lines) . " Zeilen.";
```

Ansonsten liefert PHP extrem leistungsstarke Array-Funktionen gleich mit:

**array_key_exists()**
- prüft, ob ein Schlüsel bereits in einem assoziativen Array vorhanden ist

**array_sum()**
- rechnet alle Werte eines Array zusammen

**array_unique**
- entfernt doppelte Werte aus einem Array

**in_array()**
- prüft, ob ein Element bereits in einem Array enthalten ist

Ebenfalls ansehen sollte man sich die Sortierfunktionen wie sort(), asort(), ksort() usw.

## 7.7 Variable Variablen

Kurze Erklärung: eine variable Variable ist eine solche Variable, deren Name erst zur Laufzeit ermittelt und abgefragt wird. PHP verwendet dafür die Notation ${<variablenname>}, wobei <variablenname> natürlich ebenfalls aus Variablen zusammengestellt sein kann.

```
$var1 = "Montag";
$var2 = "Dienstag";
$var3 = "Mittwoch";

for($i=1;$i<4;$i++) {
  echo "var$i = " . ${"var" . $i} . "<br>";
}
```

Variable Variablen sind beispielsweise dann besonders praktisch, wenn es darum geht array-ähnliche Aufzählungen abzufragen - beispielsweise dann, wenn die Daten aus einer Datenbank kommen und von Anwenderseite eine Eingabe gemacht werden soll:

```
<form action="script.php" method="post">   <input type="checkbox" name="sprache1"
value="1"> PHP
  <input type="checkbox" name="sprache2" value="2"> C++
  <input type="checkbox" name="sprache1" value="3"> Java
  <input type="checkbox" name="sprache1" value="4"> Perl
</form>

// Auswertung
for($i=1;$i<5;$i++) {
  if(${"sprache" . $i}=="on") {
   ...code...
  }
}
```

## 7.8 Anregungen, Tipps & Tricks

Was kann man sonst noch alles mit PHP machen? Hier dazu ein paar Ideen:

**Javascript erzeugen**

- Zum Beispiel für die Definition der Image-Objekte für MouseOver-Effekte. Macht dieses Tutorial auch - oder hattet ihr gedacht, ich würde das von Hand schreiben? :-)

**Fremde Software ansteuern**

- PHP kann COM-Objekte ansteuern. In der Windows-Welt kann damit beispielsweise ein ActiveX-Control angesprochen werden, um aus ASCII-Text PDF zu erzeugen. Oder man startet Word. Oder Outlook. Oder oder oder...

**Excel-Dateien erzeugen**

- Das ist besonders einfach: man setze mittels header()> einen Mimetype auf Excel und erzeuge eine HTML-Tabelle. Excel wird starten und die Daten korrekt anzeigen.

**Grafiken erzeugen**

- Die GD-Bibliothek stellt alle nötigen Funktionen bereit, um beispielsweise automatisch Thumbnails von hochgeladenen Bildern zu erzeugen oder Flowcharts zu generieren.

**Ungewöhnliche Dinge tun**

- Letztens habe ich eine PHP-Klasse gesehen, welche die ID3-Tags aus MP3's analysieren konnte..

## 7.9 Allgemeine Tipps

**Code dokumentieren**

- Okay, das ist kein PHP-Tipp. Und obwohl er so simpel wie selbstverständlich ist, hält sich kaum jemand daran. Wer schon einmal einen etwas komplexeren selbstgeschriebenen Code bearbeiten musste, den er drei Monate nicht angeguckt hat, weiß, wovon ich rede...

**Wiederverwertbaren Code schreiben**

- Warum immer das Rad neu erfinden? Fast immer lässt sich einmal geschriebener Code auch im nächsten Projekt verwenden. Das beste Mittel dazu sind Funktionen und Klassen.

**Funktionalität kapseln**

- Ein Informatikdozent sagte einmal: "Jedes Stück Code, das mehr als einmal aufgerufen wird, gehört in eine Funktion." Vor Jahren habe ich das belächelt - heute halte ich mich strikt daran.

**Sauber programmieren**

- Ein sehr heikles Thema. Hier muss jeder seinen Stil finden. Aber am besten ist immer noch der Stil, den auch andere sofort verstehen...

# 8 Links

Es gibt unzählige Websites zum Thema PHP. Hier eine (sicherlich subjektiv eingefärbte) Liste:

- http://www.php.net
  Die "Basis-Seite" zu PHP. Bietet Downloads aller aktuellen Versionen, Online-Handbuch und vieles mehr. Sozusagen Pflichtprogramm.

- http://www.phpwizard.net/
  PHP-Site der Maguma AG, mitbegründet von der Bozener PHP-Koryphäe Tobias Ratschiller

- http://www.phpclasses.org/
  Offene Klassenarchiv für PHP. Hier gibt es Codeschnipsel aller Art.

- http://www.phpcenter.de/
  Das deutschsprachige PHP-Portal.

- http://www.phpmag.de/
  PHP-Magazin - immer aktuelle und gute Artikel.

- http://phplib.netuse.de/
  Home of the well-known PHPLib

- http://www.phpmyadmin.net/
  phpMyAdmin ist die bekannte und leistungsstarke Frontendsoftware für mySQL

- http://www.mysql.com/
  Home of mySQL

- http://www.apache.org/
  The Apache Software Foundation

- http://selfaktuell.teamone.de/
  Umfangreiches Kompendium und Nachschlagewerk für HTML

# 9 Der Autor

Ulrich Hacke ist Jahrgang 1972 und arbeitet seit Mitte der achtziger Jahre mit Computern. Nach Abitur und Zivildienst absolvierte er das Studium der Religionspädagogik an der Ev. Fachhochschule Hannover, wo er sich in seiner Diplomarbeit intensiv mit den gesellschaftlichen Auswirkungen der Computertechnologie des Informationszeitalters beschäftigte. Es folgte ein halbes Jahr stellvertretende Projektleitung für "kirche online" in Frankfurt am Main. 1997 begann er mit der Ausbildung zum Informatikassistenten (Softwaretechnologie) am b.i.b. Hannover.

Anschließend arbeitete er haupt- und freiberuflich im IT-Umfeld und gründete 1999 mit anderen das Unternehmen TRILOS IT-Dienstleistungen, dessen Geschäftsführer er heute ist.