

Programmieren für das Internet

Martin Gasbichler Eric Knauel

12. Dezember 2005

Dieses Werk ist urheberrechtlich geschützt; alle Rechte mit Ausnahme der folgenden sind vorbehalten:

Studenten der Informatik dürfen dieses Dokument für ihre persönlichen Lehrzwecke verwenden.

Ausdrücklich verboten wird jede andere Verwendung von Ausdrucken, Dateikopien oder Paperkopien. Insbesondere darf dieses Skript nicht in irgendeiner Weise vertrieben werden und es dürfen keine Kopien der Dateien auf öffentlich zugänglichen Servern angelegt werden.

Copyright © Martin Gasbichler, Eric Knauel 2002, 2003, 2005

Inhaltsverzeichnis

4	Serverseitige Web-Programmierung	5
4.1	Common Gateway Interface	6
4.2	CGI und HTML-Formulare	9
4.3	Zustandskodierung für CGI-Programme	14
4.4	Sessionmanagement	16
4.4.1	Sessionmanagement mit Cookies	17
4.4.2	Der Cookie-Mechanismus im Detail	18
4.5	Webprogrammierung mit PHP	20
4.5.1	Datentypen in PHP	21
4.5.2	Toplevel-Syntax von PHP	21
4.5.3	Operatoren in PHP	22
4.5.4	PHP-Kontrollstrukturen	22
4.5.5	Variablen in PHP	22
4.5.6	Automatische Typkonversion	23
4.5.7	Konstanten	23
4.5.8	Funktionen	24
4.5.9	PHP-Arrays	25
4.5.10	Modulare Programmierung mit PHP	27
4.5.11	PHP und HTTP	27
4.5.12	Sessionverwaltung in PHP	28
4.5.13	PHP und HTML	29

Kapitel 4

Serverseitige Web-Programmierung

Hinter den Ressourcen, die ein Webserver für den Zugriff über HTTP zur Verfügung stellt verbergen sich, so deutet es das vorangegangene Kapitel an, Dateien im Dateisystem des Webservers. Der Pfad zu einer Ressource wird also als Pfad im Dateisystem des Webservers aufgefasst und eine `GET`-Anfrage liefert den Inhalt der Datei zurück. Man spricht in diesem Zusammenhang von *statischem Inhalt*, da der Inhalt schon vor der eigentlichen Anfrage vorhanden ist. Das Gegenstück sind Ressourcen mit *dynamischem Inhalt*, also Ressourcen deren Inhalt erst im Moment der Anfrage für diese eine Anfrage erzeugt werden.

Dynamischer Inhalt hängt in der Regel von Eingaben des Benutzers ab und wird auf Seite des Webservers von einem Programm, der *Web-Applikation*, berechnet. Grundsätzlich werden zwei technische Realisierungen unterschieden: Web-Applikationen als externe und eigenständige Programme, die über eine definierte Schnittstelle mit dem Webserver kommunizieren oder die Realisierung der Web-Applikation als eingebaute Funktion des Webservers. Dieses Kapitel stellt für beide Realisierungen Beispiele vor:

Common Gateway Interface Das *Common Gateway Interface* definiert eine Schnittstelle, um externe Programme mit der Erzeugung der Webseite zu beauftragen.

PHP Der *PHP: Hypertext Processor (PHP)* [ABD⁺05] ist ein innerhalb des Webservers realisierter Interpreter für eine Programmiersprache, die besonders auf die Programmierung von Web-Applikationen zugeschnitten ist.

Continuation-basierte Web-Programmierung Ist der Server in einer Programmiersprache geschrieben in der Continuations Werte erster Klasse sind, so kann die Web-Applikationen besonders elegant programmiert werden: Nach der Berechnung einer Seite wird die momentane Continuation eingefangen und abgespeichert. Für eine Anfrage, die eine Interaktion mit der Web-Applikation fortsetzt, ruft der Webserver die eingefangene Continuation der Web-Applikation auf.

4.1 Common Gateway Interface

Eine Web-Applikation kann mit geringem technischen Aufwand als externes Programm realisiert werden. In diesem Fall startet der Webserver das externe Programm, übergibt die Informationen, die der Client mit seiner Anfrage übermittelt hat, und reicht die durch das externe Programm berechnete Webseite an den Client weiter. Die technischen Details der Kommunikation zwischen Webserver und externem Programm regelt das *Common Gateway Interface (GGI)* [RC04].

Die technischen Mittel zur Kommunikation zwischen Webserver und dem externen Programm, dem *CGI-Programm*, sind denkbar einfach. Webserver und CGI-Programm sind über Standard-Eingabe und Standard-Ausgabe miteinander verbunden. Den Rumpf der Anfrage des Clients schreibt der Webserver in die Standard-Eingabe des CGI-Programms. Die fertig berechnete Seite gibt das CGI-Programm auf seiner Standard-Ausgabe aus, die vom Webserver gelesen wird. Andere Bestandteile der Anfrage, wie etwa die Header, werden dem Client in Umgebungsvariablen übergeben.

In dieser technischen Einfachheit der CGI-Spezifikation liegt auch die größte Stärke: Praktisch alle Webserver unterstützen CGI-Programme, eine Installation von zusätzlicher Software entfällt. Die CGI-Spezifikation ist sprachunabhängig. Jede Programmiersprache, die den Zugriff auf Standard-Eingabe, Standard-Ausgabe und auf die Umgebungsvariablen erlaubt, ist geeignet, um CGI-Programme zu schreiben.

Abbildung 4.1 zeigt, welche Vorkehrungen der Webserver trifft, bevor das eigentliche CGI-Programm ablaufen kann.¹ Für einen Prozess ist in UNIX-Betriebssystemen `exec()` die einzige Möglichkeit ein anderes Programm zu starten. Dieser Aufruf ersetzt allerdings den momentanen Prozess durch den Prozess für das zu startende Programm — `exec()` kehrt also nicht zurück. Deswegen kopiert sich der Serverprozess vor dem Aufruf von `exec()` mit `fork()` selbst und ruft `exec()` nur in dem neu entstandenen Prozess, dem Kind-Prozess, auf. Die Vorbereitungen für die Ausführung des CGI-Programms beginnen mit einem Aufruf von `pipe()`. Diese Funktion erzeugt eine *Pipe*, eine Datenstruktur zur Kommunikation zwischen Prozessen [Ste92]. Eine Pipe ist eine Verbindung zwischen zwei Prozessen, die genauso benutzt werden kann, wie eine Datei oder ein Socket. `pipe()` liefert zwei File-Deskriptoren zurück: Einen Deskriptor für den lesenden und einen für den schreibenden Zugriff auf die Pipe. In der Abbildung haben diese Deskriptoren die Nummern vier bzw. drei.

Im zweiten Schritt der Vorbereitung ruft der Server `fork()` auf, um den eigenen Prozess zu kopieren. So entsteht ein Vater- und Kind-Prozess, die über die Pipe verbunden sind. Um den CGI-Programm nicht die File-Deskriptoren mitteilen zu müssen, die für die Kommunikation mit dem Vater-Prozess benutzt werden müssen, verbiegt der Kind-Prozess die File-Deskriptoren für seine Standard-Eingabe und Standard-Ausgabe auf die Pipe. Der Kind-Prozess schließt dafür die entsprechenden File-Deskriptoren Null und Eins mittels `close()` und ruft `dup()` auf den File-Deskriptoren der Pipe auf. Durch `dup()` werden die File-Deskriptoren der Pipe kopiert — die neue Kopie hat dabei immer die niedrigst mögliche File-Deskriptor-Nummer. In diesem Fall also die eben mit `close()` geschlossenen Deskriptoren Null und Eins. Die Standard-Eingabe des

¹Der in Abbildung 4.1 gezeigte Ablauf ist vereinfacht dargestellt. In einer echten Implementierung sind zwei Pipes notwendig, da die Kommunikation über eine Pipe nur halb-duplex erfolgen kann.

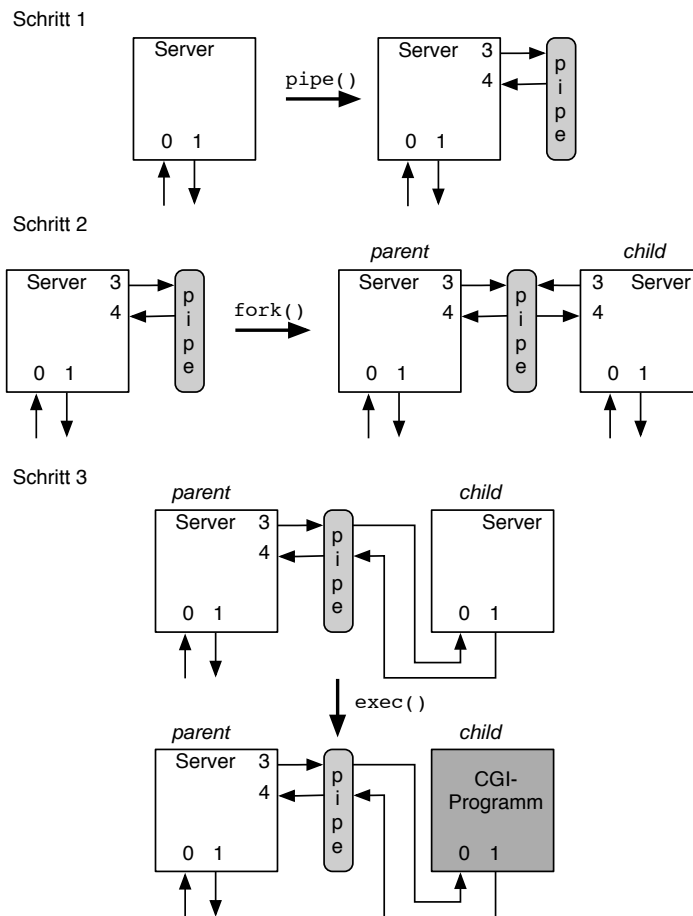


Abbildung 4.1: Aufruf eines CGI-Programms (vereinfacht)

Kind-Prozesses liest nun aus der Pipe, die Standard-Ausgabe des Kind-Prozesses erfolgt in die Pipe hinein. Im letzten Schritt ruft der Server mit `exec()` das CGI-Programm auf.

Standard-Eingabe

Der Rumpf der Anfrage-Nachricht wird vom Webserver unverändert über die Pipe an das CGI-Programm weitergereicht. Bei einer `GET`-Anfrage, die ja keinerlei Rumpf besitzt, bleibt die Standard-Eingabe leer. CGI-Programme lesen die Standard-Eingabe in der Regel nur, wenn die mit dem Aufruf des Programms verbundene `HTTP`-Anfrage mit der Methode `POST` stattfand. Die Daten, die im Rumpf übertragen werden sind in der Regel die Eingaben eines Benutzers für ein `HTML`-Formular (siehe Abschnitt 4.2).

Standard-Ausgabe

Die CGI-Spezifikation schreibt vor, daß die Ausgabe des CGI-Programms in der Standard-Ausgabe ein bestimmtes Format haben muß [RC04]. Dieses Format mit dem Namen **CGI-Response** ähnelt dem Format einer HTTP-Nachricht und besteht aus einer Reihe von Headern in RFC-882-Syntax [Res01] und einem (optionalen) Rumpf. Header und Rumpf werden durch eine Leerzeile voneinander getrennt. Ein CGI-Programm muß immer eine Ausgabe in diesem Format machen. Bleibt das CGI-Programm diese Ausgabe schuldig oder liefert dem Webserver eine syntaktisch falsche Ausgabe, antwortet der Webserver auf die Anfrage des Clients mit einem Fehler.²

Abgesehen von drei speziellen Headern, werden die Header, die das CGI-Programm ausgibt, in die Antwort-Nachricht an den Client übernommen. Die speziellen Header werden vom Webserver verarbeitet. Zu den speziellen Headern gehören:

Content-Type Dieser Header ist Pflicht und gibt den MIME-Typ des Rumpfes der Nachricht an.

Status Mit diesem Header teilt das CGI-Programm dem Webserver mit, ob es die Anfrage erfolgreich bearbeiten konnte. Argumente für diesen Header sind ein dreistelliger HTTP-Statuscode, wie Abschnitt ?? beschrieben, und eine textuelle Beschreibung des Statuscodes (entsprechend der **Reason-Phrase** in einer HTTP-Antwort-Nachricht). Dieser Header ist optional; schreibt das CGI-Programm diesen Header nicht in die Ausgabe, nimmt der Webserver den Statuscode 200 („Anfrage erfolgreich“) an.

Location Dieser Header zeigt an, daß das CGI-Programm keinen Inhalt zurückgibt, sondern eine Referenz auf eine andere Ressource. Ist das Argument des Headers eine URL, so sendet der Server eine Weiterleitung zu dieser URL an den Client. Ist das Argument ein Pfad, so verhält der Server sich so, als ob der Client ursprünglich dieses Dokument angefordert hätte.

Die Antwort-Nachricht an den Client wird vom Webserver generiert. Die Header, die dort eingehen, liest der Webserver aus der Ausgabe des CGI-Programms. Neben diesen Mechanismus definiert die CGI-Spezifikation noch eine alternative Methode zur Generierung der Antwort-Nachricht, die *Non-Parse Header*. Bei dieser Methode tritt der Webserver die Verantwortung zur Erzeugung der Antwort-Nachricht an das CGI-Programms ab und leitet dessen Ausgabe unverändert an den Client weiter.

Umgebungsvariablen

Vor der Ausführung des CGI-Programmes setzt der Webserver einige Umgebungsvariablen, um dem CGI-Programm Informationen aus der Anfrage des Clients und Informationen über den Webserver selber zu übergeben. Eine vollständige Liste der Umgebungsvariablen ist [RC04] zu entnehmen. Die folgende Aufzählung beinhaltet die in der Praxis häufig verwendeten Umgebungsvariablen:

QUERY_STRING Enthält im Fall der GET-Methode den Query-Teil der angefragten URL [BLMM94, BLFM05].

²Meist mit Statuscode 500, also interner Serverfehler.

```

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv){
    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");
    printf("<HTML><BODY>\n");
    printf("<p>Diese Seite wurde von einem CGI-Programm erzeugt</p>\n");
    printf("<p>Deine IP (oder die deines Proxys): %s</p>\n",
        getenv("REMOTE_ADDR"));
    printf("<p>Dein Browser: %s</p>\n", getenv("HTTP_USER_AGENT"));
    printf("<a href=\"%s\">Nochmal vorbeikommen</a>\n",
        getenv("SCRIPT_NAME"));
    printf("</BODY></HTML>\n\n");
    return 0;
}

```

Abbildung 4.2: Ein einfaches CGI-Programm

REQUEST_METHOD Die HTTP-Methode, mit der Client zugreifen möchte. Üblich sind GET, HEAD und POST

SCRIPT_NAME Die komplette URL, unter der das CGI-Programm zu erreichen ist

SERVER_NAME Hostname des Servers

PATH_INFO Die zusätzliche Pfadkomponenten in der URI nach dem Programmnamen

REMOTE_HOST Enthält den Hostnamen des anfragenden Hosts, wenn dieser dem Server bekannt ist. Ging die Anfrage durch einen Proxy so steht hier der Name des Proxys. Der HTTP-Header **X_FORWARDED_FOR** enthält dann eventuell den Namen des ursprünglichen Hosts.

REMOTE_ADDR Die IP-Adresse des anfragenden Hosts, siehe **REMOTE_HOST**

CONTENT_TYPE Typ des Rumpfes bei POST

CONTENT_LENGTH Länge des Rumpfes einer POST-Anfrage

Die Header der Anfrage stehen auch als Umgebungsvariablen zur Verfügung. Die einem Header entsprechende Umgebungsvariable entspricht dem Namen des Headers, um eine einen HTTP_-Präfix erweitert.

Abbildung 4.2 zeigt ein einfaches CGI-Programm, das einen Text, die IP-Adresse des anfragenden Hosts, den Browser, sowie einen Link auf sich selbst ausgibt. Abbildung 4.3 zeigt ein CGI-Programm, das alle verfügbaren Umgebungsvariablen ausdrückt.

4.2 CGI und HTML-Formulare

Web-Applikationen nehmen Eingaben des Benutzers über *HTML-Formulare* [Gro99] entgegen. Ein HTML-Formular ist ein Teil eines HTML-Dokuments, das eine

```

#include <stdio.h>

extern char **environ;

int main(int argc, char* argv[])
{
    char **p;

    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");

    printf("<html><body>\n");
    printf("<b>All environment variables:</b>\n");
    printf("<pre>\n");

    for (p = environ; *p; p++)
        printf("%s\n", *p);

    printf("</pre>\n");
    printf("</body></html>\n");
}

```

Abbildung 4.3: Ein CGI-Programm, das alle Umgebungsvariablen ausdrückt

Reihe von benannten Eingabewidgets enthält. Der Benutzer füllt das Formular aus und der Browser schickt dann die Namen der Widgets zusammen mit den Werten an den Webserver. In Regel werden die Eingaben dann serverseitig von einer Web-Applikation verarbeitet.

Ein HTML-Formular wird durch das `FORM`-Tag erzeugt. Die wichtigsten Attribute für den `FORM`-Tag sind:

- **action** Wert ist die URL an die der Browser das ausgefüllte Formular schickt
- **method** HTTP-Methode, mit der der Browser das Formular verschickt, entweder `GET` oder `POST`

Das `INPUT`-Tag erzeugt Eingabewidgets. Das Attribut `type` legt den Typ des Widgets fest. Mögliche Werte sind:

text Ein einzeliges Texteingabefeld

password Ein einzeliges Texteingabefeld, das an Stelle des Textes `*` zeigt

checkbox Eine Checkbox

radio Ein Radiobutton

reset Ein Button mit besonderer Funktion: Ein Klick auf diesen Button setzt alle Widgets des Formulars auf ihre Vorgabewerte zurück.

submit Ein spezieller Button, der den Browser dazu veranlasst, den Inhalt des Formulars an die `action`-URL zu senden.

hidden Ein unsichtbares Widget

Für jedes Widget sollte das Attribut `name` gesetzt sein, um nach der Übertragung der Formular-Daten die einzelnen Eingaben zweifelsfrei einem Widget zuordnen zu können. Das `value`-Attribut setzt bei Buttons die Beschriftung, bei allen anderen Widgets gibt es einen Default-Wert vor. Weitere Möglichkeiten, Widgets innerhalb eines FORMs zu erzeugen bieten die Tags `SELECT` für die Erzeugung von Auswahlmenüs, `TEXTAREA` für mehrzeilige Texteingabefelder und der `BUTTON`-Tag zur Erzeugung von Buttons.

Der Browser sendet den Inhalt des Formulars an die `action`-URI des Formulars, wenn der Benutzer einen Button vom Typ `submit` angeklickt. Für alle Widgets des Formulars werden Paare aus Name (gegeben durch das `name`-Attribut) und dem Inhalt des Widgets gebildet. Der Browser kodiert diese Information als `application/x-www-form-urlencoded` [BLMM94]. Das bedeutet:

1. Leerzeichen werden durch `+`-Zeichen ersetzt
2. Name und Wert werden URI-encoded [BLMM94]
3. Das Ergebnis wird in eine Zeichenkette der Form `name1=val1&name2=val2...` gebracht.

Danach entscheidet das `method`-Attribut des Formulars, wie der Browser die Information an den Server sendet. Für die `GET`-Methode hängt der Browser den String als `query`-Teil an die Anfrage an. Bei der `POST`-Methode sendet der Browser den String im Rumpf der Nachricht. Für ein CGI-Programm, das die Eingaben entgegennimmt bedeutet dies, daß die Formulareingaben einer `GET`-Anfrage aus der Umgebungsvariable `QUERY_STRING` zu lesen sind. Die Eingaben aus einer `POST`-Anfrage erhält das CGI-Programm über die Standard-Eingabe.

Formulareingaben, die mittels `GET` übertragen werden sind für den Benutzer direkt in der URL sichtbar und damit auch änderbar. In Praxis sollte deshalb darauf verzichtet werden, die `GET`-Methode für den Aufruf eines CGI-Programms mit Seiteneffekten zu verwenden.

Die `application/x-www-form-urlencoded`-Kodierung kann nur zur Kodierung von ASCII-Zeichen verwendet werden, bzw. Zeichen aus Zeichensätzen, die ein Byte pro Zeichen vorsehen. Für Eingaben in anderen Zeichensätze muß eine andere Kodierung und Methode gewählt werden. Im Zusammenspiel mit der `POST`-Methode kann durch das Attribut `enctype` des `FORM`-Tags das Kodierungsverfahren `multipart/form-data` gewählt werden. Damit ist die Übertragung von Daten, die in mehr-bytigen Zeichensätzen kodiert sind möglich.

Füllt der Benutzer zum Beispiel das folgende Formular aus

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <h1>Form-Test</h1>
    <p>Hallo, wie geht's?</p>
    <form action="http://www.uni-tuebingen.de/cgi-bin/all-env.scm"
          method="get">
      <input type="text" name="name">
      <input type="text" name="fach">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

und klickt auf den Submit-Button, so findet das CGI-Programm `all-env.scm` folgenden String in der Umgebungsvariable `QUERY_STRING`:

```
name=Martin+Gasbichler&fach=Informatik
```

Abbildung 4.4 zeigt die Funktion `query_decode`, die einen solchen String und den Namen eines Eingabewidgets als Argumente nimmt und den eingegebenen Wert für dieses Widget zurückgibt. Die Hilfsfunktion `decode_uri`, die die `pct-encoded` Zeichen (vgl. Abschnitt ??) dekodiert, ist in Abbildung 4.5 zu sehen. In der Regel überprüft ein CGI-Programm als erstes, ob alle Felder des Formulars mit sinnvollen Werten gefüllt wurden. Ist dies nicht der Fall, weist das CGI-Programm die Eingabe zurück und stellt das Formular erneut dar. Damit der Benutzer in diesem Fall nicht das komplette Formular noch einmal ausfüllen muß, ist es sinnvoll die bereits vorhandenen Werte zu übernehmen. Dies bedeutet aber, daß die HTML-Seite welche das Formular enthält auch schon berechnet werden muß. Daher ist es sinnvoll, dass ein CGI-Programm beide Aufgaben übernimmt. Das folgende CGI-Programm ist dafür ein Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include "query_decode.h"

char* validate_name(char* name)
{
    if ((name != NULL) && (strlen(name) > 3))
        return name;
    else return NULL;
}

char* validate_fach(char* fach)
{
    if (fach != NULL &&
        ((strcmp(fach, "Informatik") == 0) ||
         (strcmp(fach, "Bio-Informatik") == 0)))
        return fach;
    else return NULL;
}

void do_something(char* name, char* fach){}

int main(int argc, char** argv)
{
    char* query = getenv("QUERY_STRING");
    char* name = NULL;
    char* fach = NULL;

    if (query != NULL) {
        name = query_decode(query, "name");
        fach = query_decode(query, "fach");
    }

    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");
    printf("<HTML><BODY>\n");

    name = validate_name(name);
```

```

#include <string.h>
#include <stdlib.h>
#include "uri_decode.h"

#define BUFLEN 255

/* QUERY_DECODE()
 * Dekodiert einen in application/x-www-form-urlencoded Query-String.
 * QUERY enthaelt den Query-String, KEY den Schluessel. Es wird der
 * zum Schluessel KEY gehoerige Wert zurueckgegeben. Enthaelt der
 * Query-String keinen Schluessel names KEY gibt QUERY_DECODE() NULL
 * zurueck. */
char *query_decode(char *query, char *key)
{
    int i;
    char tmp[BUFLEN], unescaped[BUFLEN], *resbuf, *pair;

    bzero(tmp, BUFLEN);
    bzero(unescaped, BUFLEN);

    for (i = 0; i < strlen(query); i++)
        tmp[i] = (query[i] == '+') ? ' ' : query[i];

    if (decode_uri(tmp, unescaped) < 0)
        return NULL;

    for (pair = strtok(unescaped, "&"); pair; pair = strtok(NULL, "&"))
    {
        char *idx_key, thiskey[BUFLEN];
        int vallen;

        idx_key = index(pair, '=');
        if (idx_key == NULL)
            return NULL;

        bzero(thiskey, BUFLEN);
        strncpy(thiskey, pair, (int)idx_key - (int)pair);
        vallen = strlen(pair) - ((int)idx_key - (int)pair);

        if (strcmp(key, thiskey) == 0) {
            resbuf = (char *) calloc(vallen + 1, sizeof(char));
            return
                (resbuf == NULL) ? NULL : strncpy(resbuf, &idx_key[1], vallen);
        }
    }
    return NULL;
}

```

Abbildung 4.4: Dekodieren eines Query-Strings

```

fach = validate_fach(fach);

if ((name != NULL) && (fach != NULL)) {

```

```

#include "uri_decode.h"

int decode_uri(char *uri, char *resbuf)
{
    int i = 0, j = 0;
    char c, code[3] = {0, 0, 0};

    while ((c = uri[i++]) != 0) {
        if (c == '%')
            if (isxdigit(uri[i]) && (uri[i+1] != 0) && isxdigit(uri[i+1])) {
                strncpy(code, &uri[i], 2);
                resbuf[j++] = (char) strtol(code, NULL, 16);
                i += 2;
            }
            else return -1;
        else
            resbuf[j++] = c;
    }
    return 0;
}

```

Abbildung 4.5: Dekodieren eines `application/x-www-form-urlencoded`-kodierte Strings

```

do_something(name, fach);
printf("<p>Sie wurden angemeldet</p>\n");
}
else {
    printf("<FORM action=\"%s\" method=\"get\">\n",
           getenv("SCRIPT_NAME"));
    printf("<INPUT type=\"text\" value=\"%s\" name=\"name\">\n",
           (name == NULL) ? "" : name);
    printf("<INPUT type=\"text\" value=\"%s\" name=\"fach\">\n",
           (fach == NULL) ? "" : fach);
    printf("<INPUT type=\"submit\" value=\"Submit\">\n");
    printf("</FORM>\n");
}
printf("</BODY></HTML>\n\n");
return 0;
}

```

4.3 Zustandskodierung für CGI-Programme

Der Server startet ein CGI-Programm zur Bearbeitung einer einzigen Anfrage des Clients. Hat der Server die Antwort des CGI-Programms erhalten, wird das CGI-Programm nicht länger benötigt — das CGI-Programm beendet seine Ausführung. Die Ausführung des CGI-Programms ist also ausschließlich an die Beantwortung von Anfragen gekoppelt. Erstreckt sich die Interaktion zwischen Benutzer und Web-Applikation über mehrere Anfrage- und Antwort-Zyklen, so besitzt die Web-Applikation einen *Zustand*. Eine weitere Anfrage an den Server bedeutet dann eine Fortsetzung der Interaktion und letztendlich einen Über-

gang der Applikation in einen neuen Zustand. Da Web-Applikationen inhärent nebenläufig sind, können mehrere Interaktionen mit verschiedenen Benutzern nebeneinander ablaufen. Für ein CGI-Programm stellt sich nun die Frage nach einer *Zustandskodierung*: Wie ordnet das CGI-Programm eine Anfrage einer bestimmten Interaktion zu?

Um dieses Problem zu lösen, machen CGI-Programme den Zustand explizit und transportieren die Zustandsinformation von Anfrage zu Anfrage weiter. Für den Transport der Zustandsinformationen bieten sich die folgenden Techniken an:

- Browser speichern die Zustandsinformation client-seitig in *Cookies* [Coo] und transportieren die Information automatisch in den HTTP-Headern an der Server.
- Die Zustandsinformation wird im *Query*-Teil oder einem anderen Teil der URL weitergereicht. Alle Links innerhalb der Web-Applikation werden entsprechend erweitert.
- Die Zustandsinformation wird über versteckte Eingabefelder von Formular zu Formular weitergereicht.

Cookies sind eine gebräuchliche Lösung für den Transport von Zustandsinformationen. Abschnitt 4.4.1 erläutert diese Technik im Detail.

Versteckte Eingabefelder implementiert in HTML der Typ `hidden` bei den `INPUT`-Tags. Sie bieten die Möglichkeit, in einem Formular zusätzliche Informationen abzuspeichern, die beim Abschicken mitgesendet werden, aber für den Benutzer im Formular selber nicht sichtbar sind. Diese Methode wird im Folgenden nicht weiter beschrieben, da sie nur in Betracht kommt, wenn eine Web-Applikation ausschließlich über Formulare mit dem Benutzer kommuniziert.

Der Transport der Zustandsinformation als Bestandteil der URL ist eine sehr weit verbreitete Methode zur Kodierung des Zustands. Die Information muß dabei nicht unbedingt im *Query*-Teil transportiert werden, gelegentlich wird diese Information auch im Pfad der URL kodiert. Die meisten Anwendungen verwenden jedoch den *Query*-Teil, da der Zugriff auf diesen Teil innerhalb eines CGI-Programms besonders einfach ist.

Abbildung 4.6 zeigt ein CGI-Programm, das einen Zustand verwaltet. Das Programm zählt die Anzahl seiner Aufrufe durch einen Benutzer. Diese Zustandsinformation wird über den *Query*-Teil der URL weitergegeben. Das Programm gibt einen Link auf sich selbst aus und fügt den neuen Zustand (Zähler um eins erhöht) im *Query*-Teil an.

Alle genannten Methoden zur Zustandskodierung sind anfällig für Fehler und Manipulation. Deshalb muß der Entwickler bei der Implementierung von Web-Applikationen besonders sorgfältig arbeiten. Einige Fehlerquellen seien im folgenden genannt.

Cookies erfordern eine Unterstützung durch den Browser, der die Informationen verwaltet und an den Server sendet. Die meisten Browser unterstützen zwar Cookies, nicht wenige Benutzer — Schätzungen gehen von 30 Prozent aus — deaktivieren Cookies aber. Der Transport der Zustandsinformationen in der URL ist besonders manipulationsanfällig. Der Benutzer sieht die URL im Browser und kann diese leicht ändern. Die URL ist zudem schwerer lesbar. Informationen, die vom Client kommen sollten deshalb immer auf ihre Plausibilität hin untersucht werden.

```

#include <stdlib.h>
#include <stdio.h>
#include "query_decode.h"

int main(int argc, char** argv) {

    char* query = getenv("QUERY_STRING");
    char* counter_as_charp;
    int counter;

    if (query == NULL)
        counter = 0;
    else {
        counter_as_charp = query_decode(query, "counter");
        if (counter_as_charp == NULL)
            counter = 0;
        else
            counter = atoi(counter_as_charp);
    }

    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");

    printf("<HTML><BODY>");
    printf("<p>Dies ist der %d. Besuch</p>", counter);
    printf("<a href=%s?counter=%d> Nochmal vorbeikommen </a>",
           getenv("SCRIPT_NAME"), counter+1);
    printf("</BODY></HTML>");

    return 0;
}

```

Abbildung 4.6: Ein CGI-Programm mit Zustand

4.4 Sessionmanagement

HTTP bietet als Request-Reply-Protokoll keinerlei Möglichkeit, mehrere Anfragen des Clients an einen oder mehrere Server derselben Interaktion zwischen Webapplikation und Benutzer zuzuordnen. Soll zum Beispiel ein Fragebogen in mehreren Schritten, also über mehrere HTML-Seiten verteilt, realisiert werden, so wird jedesmal nachdem der Client ein Formular abgeschickt hat ein CGI-Programm angestoßen, um die Eingaben des Benutzers entgegenzunehmen, zu überprüfen, zu verarbeiten und auf dem Server abzuspeichern. Nun besteht für dieses CGI-Programm das Problem zu identifizieren, welche der ankommenden Anfragen welchen bereits gespeicherten Daten zuzuordnen sind. Für das CGI-Programm scheinen die ankommenden Daten in keinem Zusammenhang zu stehen und müssen dennoch korrekt — dies ist auch eine wichtige Frage der Sicherheit — mit bereits vorhandenen Daten eines Clients assoziiert werden.

Der Benutzer und seine Interaktion kann nicht zweifelsfrei über die IP-Adresse oder Hostnamen identifiziert werden: Es könnte sich um einen Proxy-Server handeln, oder mehrere Benutzer verwenden denselben Rechner auf dem mehrere Browser arbeiten. Die Port-Adresse ändert sich sehr wahrscheinlich von

Anfrage zu Anfrage. Auch die persistenten Verbindungen, die HTTP 1.1 vorsieht, stellen keine adäquate Lösung dar, da der Client diese Verbindungsart nicht implementieren muß.

Eine Lösung bieten die in Abschnitt 4.3 beschriebenen Techniken zum Transport von Zustand. Anstelle eines Zustands transportieren diese Mechanismen eine *Session-ID*, eine eindeutige Nummer zur Identifikation einer Interaktion. Die Daten, die ein Client bereits eingegeben hat, die für den Client berechnet wurden und überhaupt alle Daten, die für die eine Interaktion charakteristisch sind, werden als *Session-Daten* bezeichnet. Die Session-Daten werden in der Regel serverseitig gespeichert — dies sichert sie vor Manipulation durch den Benutzer. Über die Session-ID werden bei Bedarf die benötigten Session-Daten vom CGI-Programm geladen. Im Zusammenhang mit Web-Applikationen wird die Interaktion eines Benutzers mit der Web-Applikation auch als *Session* bezeichnet. Der folgende Abschnitt beschreibt, wie Sessions mit Hilfe von Cookies implementiert werden können.

4.4.1 Sessionmanagement mit Cookies

Cookies bieten dem Server die Möglichkeit, Daten in Textform auf der Clientseite abzuspeichern und später wieder zu lesen. Die Idee zur Realisierung von Cookies ist einfach: Um ein Cookie zu setzen, setzt der Server einen **Set-Cookie-Header** mit den Daten in seiner Antwort. Der Client speichert diese Daten zusammen mit dem Namen des Servers (in der Regel in einer Textdatei). Sendet der Client nun eine Anfrage, so sendet er einen **Cookie-Header** mit den gespeicherten Daten, wenn er zum Namen des Servers Cookies gespeichert hat. Die Syntax der Header ist in Abschnitt 4.4.2 beschrieben. Sessionmanagement mit Cookies realisiert man wie folgt:

1. Empfängt der Server eine Anfrage ohne **Cookie-Header**, so bedeutet dies den Beginn einer Session.
2. Der Server erzeugt eine interne Datenstruktur, um die Session zu repräsentieren.
3. Der Server erzeugt einen Schlüssel, mit dem er die neue Session eindeutig identifizieren kann, also die Session-ID.
4. Der Server sendet mit der Antwort einen **Set-Cookie-Header**, der als Daten die Session-ID enthält. Außerdem speichert der Server die Daten der Session so ab, daß er sie mit Hilfe der Session-ID wiederfinden kann.
5. Der Client speichert das Cookie und damit die Session-ID ab und schickt sie allen weiteren Anfragen im **Cookie-Header** mit.
6. Empfängt der Server nun einen **Cookie-Header**, so kann er durch die Session-ID die Daten der Session laden.
7. In den meisten Fällen bietet der Server auch eine URL an, mit der der Client die Session explizit beenden kann. Der Server schickt hier als Antwort einen **Set-Cookie-Header**, der das vorherige Cookie ungültig macht und löscht die gespeicherten Daten der Session.

Eine nicht unerhebliche Schwierigkeit besteht darin eine eindeutige, nicht vorhersehbare und nicht fälschbare Session-ID zu erzeugen. Die Session-ID kann als eine Referenz auf die Session-Daten, die serverseitig gespeichert sind, angesehen werden. Eine Alternative besteht darin, die Session-Daten direkt im Cookie zu sichern. In der Praxis wird jedoch nur noch selten von dieser Alternative Gebrauch gemacht, da dies einige Nachteile mit sich bringt: Die Daten können leichter vom Benutzer manipuliert werden und müssten bei jeder Anfrage erst einmal auf ihre Plausibilität untersucht werden. Außerdem steht zur Speicherung von Cookies in der Regel nur ein beschränkter Speicherplatz zur Verfügung. Vorteile des Session-Managements mit Cookies sind:

- Die URL enthält keinerlei Informationen über die Session und kann damit an andere Benutzer weitergegeben werden.
- Cookies sind ein erprobter Mechanismus.
- Das Programmieren mit Cookies ist einfach.

Cookies sind keine universelle Lösung für das Sessionmanagement. Unterstützt oder versteht der Browser des Clients den Header `Set-Cookie` nicht, wird er auch niemals Daten im Header `Cookie` an den Server zurückgeben. Viel häufiger ist der Fall, daß die Unterstützung von Cookies durch den Benutzer deaktiviert wurde. Dies geschieht in der Absicht, die Verfolgung der Aktivitäten „im Internet“ zu verhindern. Nach Schätzungen haben circa ein Drittel aller Benutzer Cookies deaktiviert. Es wäre darum wünschenswert ein Sessionmanagement zu haben, das unabhängig von der Cookie-Einstellung des Browsers ist.

Eine Möglichkeit dies zu realisieren ist, die Session-ID in der URL zu platzieren, beispielsweise als `PATH.INFO`. Dies hat jedoch den Nachteil, dass die Session-ID im Browser angezeigt wird und damit für andere Personen zugänglich ist. Außerdem muß dann jeder Link die Session-ID beinhalten, damit sie zu anderen Aufrufen weitergeleitet wird. Es ist aus diesen Gründen fraglich, ob diese Methode besser als der Cookie-Mechanismus ist.

4.4.2 Der Cookie-Mechanismus im Detail

Cookies sind eine Erweiterung des HTTP-Protokolls um zwei neue Header: `Set-Cookie` und `Cookie`. Ursprünglich sind Cookies auf eine Idee von Netscape [Coo] zurückzuführen. Eine kompatible Erweiterung zu der ursprünglichen Cookie-Spezifikation ist im RFC 2109 [KM97] dokumentiert, allerdings unterstützen bisher nur wenige Browser dieses RFC. RFC 2965 [KM00] ist eine überarbeitete Version des RFC 2109, aber momentan von geringer praktischer Bedeutung, da von kaum einem Browser unterstützt. Cookies im Netscape-Format sind am weitesten verbreitet, deswegen beschäftigt sich dieser Abschnitt hauptsächlich mit ihnen. Bei Netscape-Cookies wird der Syntax für den Header `Set-Cookie` durch folgende Grammatik beschrieben:

```

set-cookie   = "Set-Cookie:" cookie
cookie       = NAME "=" VALUE [";" set-cookie-av ]
NAME         = attr
VALUE        = value
set-cookie-av = [ "expires" "=" value ";" " " ]
               [ "path" "=" value ";" " " ]

```

```
[ "domain" "=" value ";" " " ]
[ "secure" ]
```

Ein **Set-Cookie-Header** ist also im Wesentlichen eine Liste von Schlüsseln und Werten. Die Schlüssel **domain**, **expires**, **path** und **secure** sind reservierte Schlüsselwörter und enthalten Verwaltungsinformationen, die für den Browser bestimmt sind. RFC 2109 definiert die zusätzlichen Verwaltungsschlüsselwörter **Comment** (Ein Kommentar zum Zweck des Cookies für den Benutzer), **Max-Age** (maximale Lebensdauer des Cookies in Sekunden ab Ausgabe) und **Version** (Versionsangabe des Cookies-Formats, momentan gültig ist Version 1). Alle Verwaltungsschlüsselwörter sind im Netscape-Format optional. Das RFC-2109-Format sieht **Version** als Pflichtfeld vor.

Die Nutzdaten der Applikation werden im Paar **NAME, VALUE** gespeichert. In einem Cookie wird genau ein solches Paar gespeichert. Um mehrere Schlüssel-Wert-Paare zu speichern, müssen mehrere **Set-Cookie-Header** gesetzt werden. Netscape empfiehlt, Werte im URL-kodierten Format zu speichern. Bei einem RFC-2109-konformen Cookie können Werte auch in Anführungszeichen gesetzt werden, müssen also nicht unbedingt kodiert werden.

domain gibt an, innerhalb welcher Internet-Domain dieses Cookie gültig ist. Der Browser liefert die Daten dieses Cookies nur an Server in dieser Domain aus. Server außerhalb dieser Domain haben keinen Zugriff auf dieses Cookie. Eine **domain** wird stets mit einem führenden Punkt angegeben.

Der Gültigkeitsbereich eines Cookies kann noch genauer durch die Angabe **path** eingeschränkt werden. Ein Cookie mit der Domain **domain=.inf.uni-tue.de** und dem **path=/shop/fanartikel** wäre zum Beispiel für die Programme <http://www1.inf.uni-tue.de/shop/fanartikel/warenkorb.cgi> und <http://www2.inf.uni-tue.de/shop/fanartikel/spezial/angebote.cgi> gültig, aber nicht für http://www1.inf.uni-tue.de/show_cart.cgi.

Der Schlüssel **expires** definiert die Lebenszeit eines Cookies. Die Zeit ist als Zeitpunkt in der Zukunft gemäß folgendem Format gegeben.

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

Dieses Format ähnelt dem aus RFC 822 [Cro82]. Liegt der Zeitpunkt in der Vergangenheit, löscht der Browser gespeicherte Cookie-Daten. Fehlt der Schlüssel **expires**, so endet die Lebenszeit des Cookies zu dem Zeitpunkt, an dem Sitzung beendet wird, also spätestens bei Beendigung des Browsers.

Enthält ein Cookie den Schlüssel **secure**, so wird dieses Cookie vom Browser übertragen, wenn die HTTP-Verbindungen mit SSL verschlüsselt ist (**https**).

Ein **Set-Cookie-Header** mit gleichem Pfad und Namen wie ein bestehendes Cookie ersetzt das bestehende Cookie. Damit ist es einem Server möglich, ein Cookie zu modifizieren: Er sendet den gleichen Pfad und Namen, aber einen anderen Wert.

Der Server kann ein Cookie nicht explizit löschen, sendet er aber einen **Set-Cookie-Header** mit gleichem Pfad und Namen, aber einem **expires**-Datum in der Vergangenheit, so wird das bestehende Cookie durch ein Cookie ersetzt, das niemals gesendet wird.

Der Browser des Clients speichert die **cookie** Daten in einer Textdatei. Kommt es nun zu einer Anfrage wird geprüft, ob diese Datei passende Cookies enthält. Cookies werden nur gesendet, wenn der Hostname der Anfrage in einer der Domains der gespeicherten Cookies liegt, die angefragte Ressource

unterhalb des Pfad-Angabe für dieses Cookie liegt und das Verfallsdatum des Cookies noch nicht erreicht ist. Die Kontrolle, welches Cookie und damit welche Daten an den Server gesendet werden, liegt damit vollständig beim Client. Entscheidet der Client, daß es ein gültiges Cookie gibt, so wird der Anfrage ein Cookie-Header mit folgender Syntax hinzugefügt:

```

cookie           = "Cookie: "
                  cookie-value *(" ; " cookie-value)
cookie-value     = NAME "=" VALUE
NAME             = attr
VALUE           = value

```

Ein Beispiel für die Abfolge von Set-Cookie- und Cookie-Headern in der Kommunikation zwischen Server und Browser. Der Server sendet:

```

Set-Cookie: customer=Eric+Knauel; path=/;
           expires=Wednesday, 15-Nov-05 23:12:40 GMT

```

Der nächste Request des Clients enthält dann:

```
Cookie: customer=Eric+Knauel
```

Angenommen, mit diesem Request wurde ein Produkt ausgewählt, dann enthält die Antwort des Servers diesen Header:

```
Set-Cookie: product_id=0042; path=/
```

Ruft der Client nun die URL `/shipping/options.html` auf, so enthält die Anfrage den Header:

```
Cookie: customer=Eric+Knauel; product_id=0042;
```

Setzt der Server nun den Header

```
Set-Cookie: shipping=express; path=/ship
```

wird der Cookie-Header bei einer Anfrage an `/shipping` und `/ships/models/` die Schlüssel `shipping`, `customer` und `product_id` enthalten. Bei einer Anfrage an `/view_order_status.cgi` enthält der Cookie-Header den Schlüssel `shipping` jedoch nicht.

Der Schutzmechanismus durch das `domain`-Feld kann ausgehöhlt werden, indem eine HTML-Seite Anfragen auf andere Domains provoziert, etwa durch das Einbinden kleiner Bilder. Damit hat auch die andere Seite eine Möglichkeit, Cookies zu setzen und zu empfangen, obwohl der Benutzer keine direkte Anfrage an sie gestellt hat. Auf Werbung im Internet spezialisierte Unternehmen verwenden diesen Mechanismus, um Informationen über Benutzer zu sammeln.

4.5 Webprogrammierung mit PHP

Die Sprache PHP [ABD⁺05] wurde speziell für die Entwicklung von Web-Applikationen entworfen. PHP steht für **PHP: Hypertext Preprocessor** — eine rekursive Abkürzung. Die Syntax der Sprache ist stark an die Syntax von C

angelehnt. PHP verfügt über eine automatische Speicherverwaltung und ein dynamisches Typsystem. Dieses Kapitel gibt einen Überblick über Syntax und Semantik von PHP und die Möglichkeiten zur Webprogrammierung mit PHP.

Das PHP speziell für die Entwicklung von Web-Applikationen gedacht ist, zeigt sich zum einen in den sprachlichen Konstrukten, aber auch in der Implementierung. PHP-Programme liegen meist im Quelltext auf dem Web-Server vor. Fordert der Client ein PHP-Programm an, startet der Web-Server einen Interpreter, der das PHP-Programm ablaufen läßt und so die eigentliche Ausgabe für den Client generiert. Der PHP-Interpreter kann auf zwei verschiedene Weisen mit dem Web-Server kommunizieren. Zum einen kann der Web-Server den Interpreter wie ein CGI-Programm starten. Wesentlich ressourcenschonender ist allerdings der Betrieb des Interpreters als Erweiterungsmodul für den Web-Server. Die meisten Web-Server erlauben es, Funktionen zur Laufzeit des Server aus Shared-object-Dateien nachzuladen. In diesem Fall teilen sich diese Funktion und der Web-Server denselben Adressraum, was wiederum eine sehr effiziente Übergabe von Daten ermöglicht. Aus diesem Grund wird PHP häufig als Modul installiert.

Der Web-Server ruft die Interpreter-Funktion nur auf, wenn die Endung der Datei im Pfad der URL `.php` lautet. Dem interpretierten Programm stehen eine Reihe von Möglichkeiten zur Verfügung, um auf die Daten der Anfrage zuzugreifen.

4.5.1 Datentypen in PHP

PHP verfügt über die folgenden skalaren Datentypen:

- Ganzzahlen (Literale: `1`, `23`, `-456`, ...)
- Fließkommazahlen (Literale: `1.2`, `3.456`, `-5.0`, ...)
- Strings (Literale: `"`, `"abc"`, `"123"`, ...). Dabei sind die üblichen Escape-Sequenzen zugelassen: `\n`, `\r`, `\o` für die oktale Notation von Zeichen (zum Beispiel `\055`) und `\xf9` für die hexadezimale Notation.
- Booleans (Literale: `TRUE`, `FALSE`)
- `NULL` Der Wert `NULL` (im Programm nicht initialisierte Variablen enthalten diesen Wert) ist von gleichnamigen Typ.

4.5.2 Toplevel-Syntax von PHP

Die oberste Ebene eines PHP-Programms besteht aus einer Folge von Statements. Ein Paar geschweifeter Klammern faßt mehrere Statements zu einem neuen Statement zusammen. Ein Strichpunkt hinter einem Ausdruck macht diesen Ausdruck zu einem Statement.

Ein PHP-Programm muß immer von einem Paar aus `<?php` und `?>` oder `<?,?>` eingeschlossen sein — nur der Code innerhalb dieser Klammern wird vom PHP-Interpreter ausgewertet (vgl. Abschnitt 4.5.13). Die Zeichen `#` und `//` leiten einen einzeiligen Kommentar ein, `/*` beginnt einen mehrzeiligen Kommentar, den `*/` wieder beendet.

4.5.3 Operatoren in PHP

PHP bietet eine große Menge von Operatoren an. Im folgenden seien nur die in der Praxis wichtigsten genannt.

arithmetisch Die arithmetischen Operatoren für die Grundrechenarten `+`, `-`, `*` arbeiten sowohl auf Fließkommazahlen als auch auf Ganzzahlen. Die Division mit `/` liefert immer eine Fließkommazahl. `%` entspricht der Modulo-Operation.

bitweise Die Operatoren `&`, `|`, `^`, `~`, `<<` und `>>` dienen der bitweisen Verknüpfung von zwei Zahlen.

vergleichend Bei den Vergleichsoperatoren gibt es neben dem Operator für strukturelle Gleichheit `==` auch noch den Operator `===`, der keine automatische Typkonvertierung vornimmt. Damit gilt zum Beispiel: `0 == FALSE`, aber nicht `0 === FALSE`.

logisch Neben den aus C bekannten logischen Operatoren `&&`, `||` und `!` gibt es auch noch `and`, `or` und `xor`.

für Strings Der Operator `.` verbindet zwei Strings.

Das PHP-Handbuch [ABD⁺05] enthält eine vollständige Liste der Operatoren und außerdem eine Tabelle, die die Präzedenzen der Operatoren angibt.

4.5.4 PHP-Kontrollstrukturen

PHP verfügt über die folgenden Kontrollstrukturen, deren Funktionsweise von anderen imperativen Programmiersprachen her bekannt ist:

- `if (test-expr) statement`
- `if (test-expr) statement else statement`
- `while (test-expr) statement`
- `do statement while (test-expr)`
- `for (init-expr ; test-expr ; end-expr) statement`

Bei den Testausdrücken ist zu beachten, daß eine Reihe von Werten als boolescher Wert `FALSE` gelten: `0`, `0.0`, `""`, `"0"`, Arrays ohne Elemente, Objekte ohne Variablen, Werte von Variablen vom Typ `NULL` (ungesetzte Variablen).

4.5.5 Variablen in PHP

Variablenamen in PHP beginnen stets mit einem `$`-Zeichen, der Rest des Namens ist case-sensitive. Es gibt in PHP keine Möglichkeit, eine Variable zu deklarieren: Die erste Benutzung (Zuweisung oder Referenzieren) führt die Variable ein. Es empfiehlt sich trotzdem, Variablen stets einen initialen Wert zuzuweisen. Einer Variablen wird wie in C mit `=` ein Wert zugewiesen:

```
$a = $b + $c - $e;
```

Wird eine Variable verwendet, ohne daß ihr zuvor ein Wert zugewiesen wurde, so hat sie den speziellen Wert `NULL` vom Typ `NULL`. Die Funktion `unset` weist einer Variable diesen Wert zu, die Funktion `is_null` gibt `TRUE` zurück, wenn eine Variable vom Wert `NULL` ist.

Der Sichtbarkeitsbereich einer Variablen entspricht dem Kontext ihrer Definition, allerdings führt — anders als in C — die geschweifte Klammer keinen neuen Kontext ein. Benützt eine Funktion zum Beispiel eine Variable zum ersten Mal, so ist die Variable bis zum Ende der Funktion definiert.

Eine Variable die auf oberster Ebene eingeführt wurde, ist innerhalb einer Funktion sichtbar, wenn die Funktion eine `global` Deklaration für diese Variable enthält:

```
$inc = 10;

function f ($x)
{
    global $inc;
    return $x + $inc;
}
```

Eine Variable hat in PHP keinen festen Typ. Im Laufe des Programms darf sich der Wert und damit auch der Typ des Wertes ändern, wie dieses Beispiel zeigt:

```
$a = 3;
$a = "a";
```

Variablen können auch innerhalb eines Strings referenziert werden:

```
$answer = 42;
$str = "Die Antwortet lautet $answer";
```

Die alternative Syntax für Stringlitterale bei der der String durch zwei Hochkommata (') eingeschlossen ist, unterstützt dies nicht.

4.5.6 Automatische Typkonversion

Viele PHP-Operatoren vollführen eine *automatische Typkonversion*, wenn die Argumente nicht vom erwarteten Typ sind. Arithmetische Operatoren wie `+` konvertieren ihre Argumente in Zahlen, `if` konvertiert den Wert des Testausdrucks in einen booleschen Wert, `print` konvertiert sein Argument in einen String. Abbildung 4.7 gibt einen groben Überblick über die Regeln, die zur Konvertierung von Typen verwendet werden. Der Programmierer kann Typkonversion auch erzwingen, indem er einen C-typischen *Type-Cast* anwendet.

4.5.7 Konstanten

Die Spezialform `define(name, expr)`; definiert die Konstante *Name* mit dem Wert des Ausdrucks *expr*. Im Unterschied zu Variablennamen, müssen die Namen für Konstanten nicht mit einem Dollarzeichen beginnen. Konstanten sind außerdem *superglobal*. Dies bedeutet die Konstante ist global sichtbar und Funktionen können auf die Konstante zugreifen ohne diese vorher mit `global` in den eigenen Sichtbarkeitsbereich zu importieren. Beispiel:

Zahl → String	Gemäß Darstellung
Fließkomma → Ganzzahl	Durch Rundung
String → Zahl	Die (dezimalen) Ziffern am Anfang des Strings werden gelesen und in eine Zahl umgewandelt. Ist diese Umwandlung nicht möglich, wird der String zum Wert 0 umgewandelt.
Boolean → A	Booleans werden in Zahlen umgewandelt, wobei TRUE → 1 und FALSE → 0 gilt. Die Zahl wird dann in den Typ A konvertiert.
A → Boolean	Der Typ A wird zunächst eine Zahl konvertiert, die dann in einen Boolean-Wert umgewandelt wird. Es gilt: 0.0 → FALSE, 0 → FALSE und 1 → TRUE.
String → Boolean	Für die Umwandlung eines Strings in einen Boolean gilt: "" → FALSE, "0" → FALSE, "0.0" → TRUE. Andere Strings werden nach TRUE konvertiert.
NULL → Boolean	Liefert FALSE
NULL → Ganzzahl	Liefert 0
NULL → String	Liefert den leeren String

Abbildung 4.7: Regeln für die automatische Typkonversion

```
define(ABC, 123);

function f ()
{
    define(XYZ, 456);
    return ABC + XYZ;
}

echo f();
echo XYZ;
?>
```

4.5.8 Funktionen

Das Schlüsselwort `function` leitet eine Funktionsdeklaration ein. Es folgt der Name der Funktion und eine Liste der Funktionsparameter in Klammern:

```
function f ($param1, $param2)
```

Danach folgt der Rumpf in geschweiften Klammern:

```
{
    <statement> ...
}
```

Das Schlüsselwort `return` erlaubt es einer Funktion, einen Wert zum Aufrufer zurückzugeben:

```
return <expr>;
```

Eine Funktion wird aufgerufen, indem ihr Name gefolgt von einer Liste der Parameter in Klammern angegeben wird:

```
f(23, 42);
```

Die Parameterübergabe ist standardmäßig *call-by-value*, wird vor den Parameternamen ein `&` eingefügt, so ändert sich dies zu *call-by-reference*:

```
function f ($p1){
    $p1[0] = "c";
    print $p1[0];
}
```

```
function f_ref (&$p1){
    $p1[0] = "d";
    print $p1[0];
}
```

```
$a = "aaaa";
f($a);           // druckt "c"
print $a[0];    // druckt "a"
f_ref($a);      // druckt "d"
print $a[0];    // druckt "d"
```

Außerdem ist es möglich, einen Vorgabewert für Parameter anzugeben, indem man ein *=const-expr* an die Parameterdeklaration anhängt:

```
function f ($a, $b = 3){
    return $a + $b;
}
```

```
print (f(5));
```

4.5.9 PHP-Arrays

In PHP stehen folgende nicht-primitive Datentypen zur Verfügung:

- Arrays (Literele: `array()`, `array(1,2)`, `array("a",12)`, `array("de" => "Deutschland", "us" => "United States")`)
- Klassen

Arrays sind die bei weitem am häufigsten verwendete Datenstruktur in PHP. PHP-Arrays verhalten sich in einigen Punkten anders als die aus C bekannten Arrays:

- Die Größe eines Arrays ist nicht fix. Es können weitere Elemente eingefügt werden. Die momentane Größe eines Arrays liefert die eingebaute Funktion `count`. Allerdings ist diese Angabe nicht unbedingt gleichbedeutend mit dem höchsten Index, da das Array „Löcher“ haben kann.
- Der Index eines Arrays kann ein beliebiger Wert sein. Wenn ein Element ohne die Angabe eines Indexes hinzugefügt wird, wird die nächste freie Zahl als Index genommen.

- Ein ungültiger Index beim Lesen führt nicht zu einem Fehler sondern liefert NULL zurück.
- Da PHP eine dynamisch getypte Sprache ist, müssen die Elemente eines Arrays nicht vom gleichen Typ sein.

Die Zuweisung und das Dereferenzieren eines Array-Feldes erfolgt mit der aus C bekannten []-Notation:

```
$a[0] = 3;
$a["one"] = 4;
```

```
echo $a[0];
echo $a["one"];
```

Über Arrays, die ausschließlich mit Zahlen indiziert sind, kann der Programmierer mit einer `for`-Schleife iterieren:

```
$arr = array ("a", "b", "c");
$arr[] = "d";
$arr[4] = "e";
```

```
$num_elements = count ($arr);
print ("\n$num_elements\n");
```

```
for ($idx = 0; $idx < $num_elements; $idx = $idx + 1) {
    echo "$arr[$idx]\n";
}
```

Ist das Array jedoch auch mit Werten anderen Typs indiziert, wird mit dem `foreach`-Konstrukt über alle Einträge iteriert:

```
foreach(array-exp as var) statement
```

Dies bindet die Variable *var* nacheinander an jeden Wert im Array *array-exp* und wertet dann das Statement *statement* aus:

```
$arr=array("erstes","zweites");
$arr["bla"]="bei bla";
$arr[]="drittes";
```

```
foreach ($arr as $inhalt)
    print ("$inhalt\n");
```

Die erweiterte Syntax

```
foreach(array-exp as var1 => var2) statement
```

bindet die Variable *var1* an jeden Index und die Variable *var2* an jeden Wert:

```
$arr=array("erstes","zweites");
$arr["bla"]="bei bla";
$arr[]="drittes";
```

```
$num_elements = count ($arr);
```

```
foreach ($arr as $index => $inhalt)
    print ("$index points to $inhalt\n");
```

Da ein Element eines Arrays wieder ein Array sein kann ist es leicht möglich, mehrdimensionale Arrays zu erstellen:

```
$continents = array ("Europe" => array ("de", "uk"),
    "America" => array ("us", "ca"));

echo ($continents["Europe"][1]); // prints "uk"
echo ($continents["America"][1]); // prints "ca"
```

Der `+`-Operator hängt zwei Arrays zu einem zusammen. Die Funktion `unset` löscht ein Feld aus einem Array; das Prädikat `isset` gibt an, ob ein Feld an einem gegebenen Index gesetzt ist.

4.5.10 Modulare Programmierung mit PHP

PHP verfügt über kein Modulsystem. Stattdessen bietet es lediglich die Möglichkeit, andere Quelldateien einzubinden. Dies geschieht entweder mit (`require filename-expr`) oder mit (`include filename-expr`). *filename-expr* muß dabei zu einem String evaluieren, der als Dateiname interpretiert wird. Diese Datei wird dann innerhalb des Suchpfades, der in der Option `include_path` des Webservers steht, gesucht. Der Unterschied zwischen `require` und `include` besteht lediglich darin, daß `require` das Programm abbricht, wenn es die Datei nicht finden kann, während `include` in diesem Fall nur eine Warnung ausgibt.

Die Variationen `require_once` und `include_once` sorgen dafür, daß die angegebenen Dateien während einer Ausführung des Programms nur genau einmal eingebunden werden.

4.5.11 PHP und HTTP

Die Informationen aus der Anfrage des Clients stehen dem PHP-Programm in einigen vordefinierten superglobalen Arrays zur Verfügung. Die folgende Auflistung umfasst die wichtigsten vordefinierten Arrays:

`$_COOKIE` Array mit allen Cookies, die in der Anfrage enthalten waren. Der Name des Cookies dient hierbei als Schlüssel.

`$_GET` Array mit allen Werten, die im `QUERY_STRING` einer `GET`-Anfrage übermittelt wurden. Die Namen der Eingabefelder bilden die Schlüssel für das Array.

`$_POST` Array mit allen verfügbaren Werten aus dem Rumpf einer `POST`-Anfrage.

`$_SESSION` Ein Array, das die Session-Daten enthält. Vergleiche Abschnitt [4.5.12](#).

`$_REQUEST` Feld mit allen verfügbaren Werten aus `$_GET`, `$_POST`, und `$_COOKIE`.

Mit Hilfe dieser Felder ist es leicht möglich, die mit der HTTP-Anfrage übertragenen Werte an Variablen zu binden:

```
$uname = $_POST['username'];
$pw = $_REQUEST['password'];
$date = $_REQUEST['date'];
```

Die Konfigurationsoption `register-globals=on` konfiguriert den PHP-Interpreter so, daß sämtliche Informationen der Anfrage als gleichnamige globale Variablen zur Verfügung stehen. Im obigen Beispiel gäbe es also die Variablen `$username`, `$password` und `$date`. Da dies ein Sicherheitsrisiko darstellen kann (eine fingierte URL kann bestehende Variablen überschreiben) ist diese Option seit PHP 4.2 standardmäßig abgeschaltet.

Eine Alternative zur automatischen Einbindung *aller* Variablen bietet die Funktion `import_request_variables` (*types, prefix*). Sie macht die Variablen aus *types* (ein String mit einem oder mehreren der Buchstaben *g*, *p* und *c*, die für die Felder `GET`, `POST` und `Cookies` der Anfrage stehen) mit dem Präfix *prefix* als globale Variablen verfügbar.

Die Funktion `header` fügt der Antwort des Web-Servers einen weiteren Header hinzu oder ersetzt einen Header. Die `header`-Funktion sollte allerdings nur aufgerufen werden, wenn die Ausgabe des Rumpfes der HTTP-Nachricht noch nicht begonnen hat. Andernfalls gibt `header` eine Warnung aus. Das Prädikat `headers_sent` zeigt an, ob die Header einer HTTP-Antwort bereits gesendet wurden und `header` somit nicht mehr angewendet werden sollte. Mit `header` kann auch direkt die Status-Zeile gesetzt werden:

```
header("Status: 404 Not Found");
```

Um Cookies zu setzen ist es nicht nötig, den `Set-Cookie`-Header selbst zusammenzustellen: Die Funktion `setcookie` erzeugt den erforderlichen Header. Die Argumente sind:

```
setcookie(string name [,string value [,int expire
           [,string path ,string domain ,int secure]])
```

Die Funktion `setcookie` kodiert das `value`-Argument automatisch; im `$_COOKIES`-Feld steht der URI-dekodierte Wert. Beispiel:

```
if (isset($_COOKIE["login"]))
print ("Welcome back, " .($_COOKIE['login']));
else
setcookie("login","anonymous user");
print ("You logged in as an anonymous user");
```

4.5.12 Sessionverwaltung in PHP

Über die superglobale Variable `$_SESSION` bietet PHP eine eingebaute Unterstützung für Sessionmanagement. PHP speichert alle Felder dieses Arrays am Ende des PHP-Programms in einer Datei auf dem Server ab. Beim nächsten Start lädt der Interpreter diese Datei wieder und restauriert das Array.

Das PHP-Programm muß die Funktion `session_start` einmal aufrufen, um das Session-Management zu aktivieren. Die zugehörige Session-ID verwaltet PHP entweder in einem Cookie oder im `query`-Teil der URL, falls der Benutzer Cookies deaktiviert hat. Dabei kümmert sich PHP selbst darum, daß alle relativen URLs den Zusatz im `query`-Teil bekommen. Beispiel:

```

<?php
session_start();

if (!(isset ($_SESSION['count'])))
    $_SESSION['count'] = 1;
else
    $_SESSION['count'] += 1;

$count = $_SESSION['count'];
?>
Dies ist Ihr <? echo $count; ?>. Besuch<p>

Nochmal <A HREF=<? echo $_SERVER['SCRIPT_NAME']; ?>>vorbeikommen</A>

```

Ist `register-globals` eingeschaltet, so werden die Felder aus `$_SESSION` automatisch zu globalen Variablen. Sie können dann mit den Funktionen `session_register`, `session_is_registered` und `session_unregister` manipuliert werden.

Für das Abspeichern des `$_SESSION`-Arrays kommt noch ein weiteres wichtiges Feature ins Spiel: Mit der Funktion `serialize` ist es möglich, die Stringrepräsentation eines Wertes zu erzeugen. Die Funktion `unserialize` nimmt einen solchen String als Argument und erzeugt daraus wieder den ursprünglichen Wert. Die Funktion `session_serialize` wendet diese Funktionen automatisch an — der PHP-Interpreter ruft sie am Programmende automatisch auf, wenn das Sessionmanagement gestartet wurde.

4.5.13 PHP und HTML

In einer `.php`-Datei kann HTML mit PHP vermischt werden: Daten, die nicht von einem `?php`-Tag eingeschlossen sind, werden direkt an den Client geschickt. Dabei ist es nicht notwendig, daß innerhalb der `?php`-Tags für sich allein gültiger PHP-Code steht. Es ist auch möglich, den Code erst in weiteren `?php`-Tags zu vervollständigen. Damit ist es zum Beispiel möglich, die schließende Klammer einer Statementgruppe in einem anderen `?php`-Tag unterzubringen und dazwischen HTML-Code einzufügen. Zehn Zeilenumbrüche erzeugt man mit dieser Methode dann so:

```

<?php for ($i=0; $i<10; $i++) { ?>
<br>
<?php } ?>

```

Für alle Kontrollstrukturen gibt es eine alternative Syntax. Sie entsteht aus der ursprünglichen Syntax, indem die öffnende Klammer durch einen Doppelpunkt und die schließende Klammer durch ein `endname` ersetzt wird. Damit sieht obiges Beispiel dann so aus:

```

<?php for ($i=0; $i<10; $i++): ?>
<BR>
<?php endfor; ?>

```

Haben mehrere HTML-Eingabewidgets den gleichen Namen und endet dieser mit `[]`, so ist es möglich, die Werte der Eingabefelder als ein Array in PHP

```

<?php
$menu = array("Pizza", "Lasagne", "Spaghetti", "Panna Cotta");

if ($_GET['action']=="submitted"):
    echo "Ihre Bestellung:<BR>";

    $client = $_GET['personal'];
    echo "Name:", $client["name"],"<BR>";
    echo "Straße:", $client["street"],"<BR>";

    $dishes = $_GET['dishes'];
    for($i = 0; $i < count($dishes); $i++)
        echo $menu[$dishes[$i]], "<BR>";

else : ?>

<FORM action=<?=$_SERVER['SCRIPT_NAME']?> method="get">

Name: <INPUT type="text" name="personal[name]"><br>
Straße: <INPUT type="text" name="personal[street]"><br>

<P> Bitte Gerichte auswählen: </P>
<SELECT multiple name="dishes[]">

<?php for($i=0;$i<count($menu);$i++): ?>

<OPTION value=<?=$i?>> <?=$menu[$i]?> </OPTION>

<?php endfor; ?>

</SELECT><BR>
<INPUT type="hidden" name="action" value="submitted">
<INPUT type="submit" name="submit" value="Abschicken!">
</FORM>

<?php endif; ?>

```

Abbildung 4.8: Eingabewidgets und Arrays in PHP

verfügbar zu machen. Damit ist es leicht möglich, ein dynamisch erzeugtes Formular einzulesen, wie es in [Abbildung 4.8](#) zu sehen ist.

Literaturverzeichnis

- [ABD⁺05] Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Philip Olson, Georg Richter, Damien Seguy, and Jakub Vrana. Php manual, October 2005. 5, 20, 22
- [BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005. 8
- [BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard), December 1994. Updated by RFCs 1808, 2368, 2396, 3986. 8, 11
- [Coo] Persistent Client State HTTP Cookies. http://wp.netscape.com/newsref/std/cookie_spec.html. 15, 18
- [Cro82] D. Crocker. Standard for the format of ARPA Internet text messages. RFC 822 (Standard), August 1982. Obsoleted by RFC 2822, updated by RFCs 1123, 1138, 1148, 1327, 2156. 19
- [Gro99] W3C HTML Working Group. Html 4.01 specification, December 1999. 9
- [KM97] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2109 (Proposed Standard), February 1997. Obsoleted by RFC 2965. 18
- [KM00] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2965 (Proposed Standard), October 2000. 18
- [RC04] D. Robinson and K. Coar. The Common Gateway Interface (CGI) Version 1.1. RFC 3875 (Informational), October 2004. 6, 8
- [Res01] Peter W. Resnick. Internet message format. <http://www.faqs.org/rfcs/rfc2822.html>, April 2001. 8
- [Ste92] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley Publishing Company, Inc., 1992. 6

Index

- CGI-Response, 8
- CONTENT_LENGTH, 9
- CONTENT_TYPE, 9
- Content-Type, 8
- Cookie-Header, 17
- FORM, 10, 11
- INPUT, 15
- Location, 8
- PATH_INFO, 9
- QUERY_STRING, 8
- Query-Teil, 15
- REMOTE_ADDR, 9
- REMOTE_HOST, 9
- REQUEST_METHOD, 9
- SCRIPT_NAME, 9
- SERVER_NAME, 9
- Set-Cookie-Header, 17
- Status, 8
- X_FORWARDED_FOR, 9
- close(), 6
- dup(), 6
- enctype, 11
- exec(), 6, 7
- fork(), 6
- hidden, 15
- multipart/form-data, 11
- pipe(), 6
- serialize, 29
- unserialize, 29

- automatische Typkonversion, 23

- CGI, 5, 6
- CGI-Programm, 6
- Common Gateway Interface, 5, 6
- Continuation, 5
- Cookies, 15

- dynamischer Inhalt, 5

- Eingabewidget, 10

- HTML-Formular, 7, 9

- Interpreter, 5

- MIME, 8

- Non-Parse Header, 8

- PHP, 5
- PHP: Hypertext Processor, 5
- Pipe, 6
- Programmiersprache, 5

- Session, 17
- Session-Daten, 17
- Session-ID, 17
- Shared-object-Dateien, 21
- statischer Inhalt, 5
- Statuscode, 8
- superglobal, 23

- Type-Cast, 23

- Web-Applikation, 5

- Zustand, 14
- Zustandskodierung, 15