1.9 HTML/CGI

1.9.1 Was passiert mit den Formulardaten?

Im folgenden Beispiel soll nachgesehen werden, was mit den abgeschickten Formulardaten passiert. Hier ein einfaches HTML-Dokument (`sende_bsp.html`):

```
1 <html><head></head><body>
2   <form action="http://127.0.0.1/cgi-bin/getenv.sh">
3     Meldung:<input type="text" name="meldung" /><br />
4     <input type="submit" /><br />
5   </form>
6 </body></html>
```

In Zeile 2 ist im `action`-Attribut diesmal keine E-Mail-Adresse, sondern eine URL mit HTTP als Protokoll angegeben. Die Formulardaten sollen also an einen HTTP-Server gesandt werden. Nun gibt man einen Text ein (Abbildung 1). Sobald man den Submit-Knopf drückt, schickt der

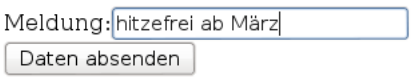

Abbildung 1: Eingabedaten im Browser

Browser eine Anfrage an den genannten Server (127.0.0.1), die ungefähr wie folgt aussieht:

```
GET /cgi-bin/getenv.sh?meldung=hitzefrei+ab+M%E4rz HTTP/1.0
(Leerzeile)
```

Im Browser wird eine daraufhin eine neue Seite geöffnet; die neue Adresszeile sieht entsprechend aus (Abbildung 2). Wie man sieht, wurden die Daten einfach an die im `action`-Attribut angegebene

Abbildung 2: Adresszeile

URL angehängt (abgetrennt durch ein Fragezeichen). Die Leerzeichen im Text wurden durch Pluszeichen ersetzt und das „ä“ durch eine Zeichenkombination.

1.9.2 Wie reagiert der Server?

Was auf der Seite des Servers passiert, hängt natürlich vom Server und seiner Einrichtung ab. In diesem Fall soll auf dem Server-Rechner ein Web-Server (Apache 2) installiert sein. Er kann die Anfrage entgegennehmen. Er leitet sie an ein Programm weiter, an ein so genanntes CGI-Skript. Dieses Programm könnte die Eingabedaten auswerten. Anschließend erzeugt das Programm eine Ausgabe, die aus einem HTTP-Kopf und einem HTML-Dokument (oder einem Text-Dokument) besteht, z. B. so:

```
HTTP/1.0 200 OK
Content-Type: text/html

<html><head></head><body>
    Ihre Meldung wird bearbeitet, vielen Dank!
</body></html>
```

Die erste Zeile (Statuszeile) wird vom Webserver hinzugefügt, die anderen Zeilen musste das Programm selbst schreiben, und zwar mit demselben Befehl, der in der entsprechenden Sprache für die Textausgabe auf dem Bildschirm benutzt wird (`antwort.sh`):

```
#!/bin/bash
echo "Content-Type: text/html"
echo ""
echo "<html><head></head><body>"
echo "    Ihre Meldung wird bearbeitet , vielen Dank! "
echo "</body></html>"
```

In diesem Beispiel wird allerdings nur eine Meldung ausgegeben, ohne dass die Formulardaten wirklich verarbeitet wurden. Dazu müsste man nämlich erst einmal wissen, wie man an diese Daten herankommt.

1.9.3 Wie kommt das CGI-Skript an die Benutzereingaben?

Das CGI-Skript bekommt die Formulardaten im Inhalt der Umgebungsvariablen `QUERY_STRING` zur Verfügung gestellt. Im obigen Beispiel lautet der Inhalt von `QUERY_STRING`:

```
meldung=hitzefrei+ab+M%E4rz
```

Falls man ein Formular mit mehreren Feldern hat, muss man dieses Inhalt noch fachgerecht zerlegen.

1.9.4 Beispiel für die Umgebungsvariablen eines CGI-Skripts

Hier ist ein CGI-Skript, das alle seine Umgebungsvariablen auflistet (`getenv.sh`):

```
#!/bin/bash
echo "Content-Type: text/plain"
echo ""
env
```

Die Datei kann muss man nun an die richtige Stelle im Dateisystem kopieren:

Terminal

```
root@debian964:~# cp getenv*.sh /usr/lib/cgi-bin
```

Wenn man jetzt mit dem obigen HTML-Dokument dieses Skript aufruft, bekommt man eine ganze Reihe von Umgebungsvariablen angezeigt:

```
SERVER_PORT=80
HTTP_HOST=127.0.0.1
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.sh
REMOTE_ADDR=127.0.0.1
... (viele weitere) ...
QUERY_STRING=meldung=hitzefrei+ab+M%E4rz
REQUEST_METHOD=GET
```

Jetzt kann man sich als Skript aus diesen Variablen bedienen. Man kann etwa die Inhalte in eine Datei schreiben, die jemand anders auslesen soll, man kann sie als Mail verschicken oder man kann damit auf eine Datenbank zugreifen, die auf dem Server-Rechner läuft.

1.9.5 Die POST-Nachricht für große Datenmengen

Durch die Umgebungsvariable ist die Größe der Eingabedaten schon vom Prinzip her beschränkt. Für alle Anwendungen, bei denen das nicht sinnvoll ist, braucht man einen anderen Weg.

Im Dokument `sende_bsp_post.html` ist nur eine Zeile anders als im bisherigen Beispiel:

```
<form action="http://127.0.0.1/cgi-bin/getenvpost.sh" method="post">
```

Das Attribut method="post" legt fest, dass der Browser an den Server keine GET-, sondern eine POST-Nachricht schickt. Das bewirkt, dass die Eingabedaten nicht mehr in der Adresszeile weitergegeben werden, sondern im Rumpf der POST-Nachricht[1]:

```
POST /cgi-bin/getenvpost.sh HTTP/1.0
Content-Length: 27

meldung=hitzefrei+ab+M%E4rz
```

Das neue CGI-Skript muss die Eingabe jetzt anders verarbeiten als bisher (`getenvpost.sh`):

```
1  #!/bin/bash
2  echo "Content-Type: text/plain"
3  echo ""
4  env
5  echo "--- POST-DATEN: ---------"
6  read x
7  while [ "$x" != "" ]
8  do
9      echo "$x"
10     read x
11 done
```

Wenn man jetzt mit dem neuen Dokument das neue Skript aufruft, ist die Umgebungsvariable `QUERY_STRING` leer:

```
SERVER_PORT=80
HTTP_HOST=127.0.0.1
...
CONTENT_LENGTH=27
QUERY_STRING=
CONTENT_TYPE=application/x-www-form-urlencoded
REQUEST_METHOD=POST
```

Stattdessen sind die Länge der Eingabe (`CONTENT_LENGTH`) und ihr MIME-Type angegeben (`CONTENT_TYPE`). Oft ist es Geschmackssache, ob man die GET- oder die POST-Nachricht verwendet.

[1] eine GET-Nachricht könnte das nicht; sie hat nämlich gar keinen Rumpf