Das einzige deutschsprachige Linux-Magazin

LINUX MAGAZIN 1/98

ISSN 1432 - 640 X DM 9,- ÖS 75,- SFR 9,-

LINUX MAGAZIN

1/98

Programme beobachten mit dem Syslog Dämon

Arbeitsschritte und Zeiten protokollieren

Linux News

Neues aus dem Internet

Testbericht

Red Hat CDE

Lynx

Mit ASCII-Power ins Web

AC3D

Universeller 3D-Modellierer

Perl Snapshot

Eingebettet

von Michael Schilli

Auf dem Comprehensive Perl Archive Network (CPAN) lagert eine schier unglaubliche Fülle an Perl-Modulen. Was man mit ihnen alles anstellen kann, zeigt unsere Perl-Reihe. In der heutigen Folge geht es um eine der vielen Möglichkeiten Perl in HTML-Seiten einzubetten.

CGI-Skripts in Perl, die HTML über *print*-Anweisungen ausspucken, bereiten mir Kopfweh. Heute zeige ich Alternativen anhand eines *Rotating Banners* - eines dieser (natürlich wegen kaptitalistischer Unterwanderung des Internet Yada Yada Yada aufs Schärfste zu verurteilenden) Werbe-Bildchen, die sich bei jedem neuen Aufruf einer Webseite ändern.

Die Funktion *time* liefert in Perl bekanntlich die Anzahl der vergangenen Sekunden seit dem 1.1.1970, also ergibt *time % 4* eine für CGI-Skripts akzeptable Streuung zwischen *0* und *3* im Sekundentakt. Das kurze Skript

```
#!/usr/bin/perl -w

print „Content-Type: text/html\n\n“;
print „<HTML><TITLE>Titel</TITLE>\n“;
print „<BODY>\n“;
print „<IMG SRC=\“/banners/banner“,
          time % 4 + 1, „.jpg\“„;
print „</BODY></HTML>\n“;
```

produziert demnach eine einfache Webseite, die - mehr oder minder zufällig - eines der im */banners*-Verzeichnis vordefinierten Bilder *banner1.jpg - banner4.jpg* enthält. Funktionieren tut's, aber wie häßlich ist der Code!

Als Alternative läßt das in dieser Reihe schon mehrfach verwendete *CGI*-Modul von *Lincoln Stein* das ganze schon wieder mehr nach *Perl* aussehen:

```
#!/usr/bin/perl -w

use CGI;
$q = new CGI;
print $q->header();
print $q->start_html(„Titel“);
print $q->img({src => „/banners/banner“ .
                                (time % 4 + 1) .
„.jpg“});
print $q->end_html();
```

Enthält die HTML-Seite aber außer dem Banner noch allerlei graphischen Schnickschnack und Text, wird auch diese Lösung irgendwann unhandlich.

Und nun zu etwas *gaaanz* :-) anderem: Besteht die Ausgabe eines CGI-Skripts hauptsächlich aus statischem HTML mit nur wenigen dynamischen Daten, liegt der Ansatz nahe, HTML zu schreiben, das in speziellen Tags Perl-Code ausführt:

```
<HTML>
  <TITLE>Titel</TITLE>
  <BODY>
    <IMG SRC=“/banners/banner[+ time % 4 + 1
+].jpg“>
  </BODY>
<HTML>
```

Der offensichtliche Vorteil des Verfahrens: Diesen ``Code'' schluckt selbst ein HTML-Editor, mit dem auch ein Perl-unkundiger Layouter die Seite noch nach Herzenslust um statische Komponenten erweitern kann.

Mit einer derartigen Web-Page kann natürlich ein Browser, der kein Perl versteht, nichts anfangen. Das Geheimnis: Serverseitig werden bei jedem ankommenden Request die eingebetteten Perl-Snippets ausgeführt und an Stelle der dem Browser unverständlichen Tags dynamisch erzeugter Text eingefügt.

Das Paket *Embperl* von Gerald Richter erledigt genau das. Zwei Möglichkeiten gibt's: Entweder ein CGI-Script arbeitet den eingebetteten Perl-Code ab - oder der Server veranlaßt dies (muß in diesem Fall Apache sein).

Aufruf als CGI

Hierzu muß das CGI-Skript *embpexec.pl* aus dem *Embperl*-Paket im *cgi-bin*-Pfad des Web-Servers vorliegen (siehe Kasten *Installation*). Heißt die HTML-Seite mit dem eingebetteten Perl-Code *banner.html* und liegt direkt im *htdocs*-Verzeichnis des Web-Servers, liefert eine Anfrage nach dem URL

```
http://my.host.com/cgi-bin/embpexec.pl/
banner.html
```

eine Seite mit dynamisch generiertem Text zurück. Der Server reicht dabei die Pfadangabe */banner.html* an das Skript *embpexec.pl* weiter, welches sich wiederum die Datei *banner.html* schnappt und

- zwischen *[+ ... +]* liegenden Code ausführt, wobei es dieses *Embperl*-spezifische Tag durch den Rückgabewert des Snippets ersetzt
- zwischen *[- ... -]* liegenden Code *stillschweigend* ausführt (schreibt das Snippet nach STDOUT, gelangt die Ausgabe dennoch in die Seite)
- zwischen *[$ ... $]* liegenden Code als *Embperl*-Meta-Kommando interpretiert, und es so erlaubt, mit *if* und *while*-Konstrukten Teile der HTML-Seite *bedingt* abzuarbeiten:

```
[$ while (...) $]
...
[$ endwhile $]
[$ if (...) $]
...
[$ elsif (...) $]
...
[$ else $]
...
[$ endif $]
```

Stehen zwei Banner mit den Dateinamen *first.jpg* und *second.jpg* zur Verfügung, bringt folgender HTML-Text zufällig jeweils eines von beiden zum Vorschein:

```
[- @banners = („first.jpg“, „second.jpg“); -]
[- $banner = $banners[time % 2]; -]
<IMG SRC=“/banners/[+ $banner; +]“>
```

Letzteres Problem hätte sich auch mittels einer *if*-Bedingung aus dem *Embperl*-Metakommando-Fundus lösen lassen:

```
[$ if (time % 2) $]
    <IMG SRC=“/banners/first.jpg“>
[$ else $]
    <IMG SRC=“/banners/second.jpg“>
[$ endif $]
```

Wie man drei Banner aus einer Serie von sechsen auf einer Seite nebeneinander darstellt, ohne daß Duplikate auftreten, zeigt die HTML-Datei aus Listing 1. Jedes dargestellte Bild verfügt zusätzlich über einen Hyperlink, der den an der Werbung interessierten Anwender per Mausklick auf die verknüpfte Webseite beamt.

Das erste Perl-Snippet, das sich in Listing 1 über die Zeilen 2 bis 11 erstreckt, initialisiert den Zufallsgenerator mit *srand()* und der aktuellen Uhrzeit in Sekunden als *Seed*. Ohne diese Aktion brächte jeder Aufruf die gleichen Zufallswerte daher, da jedes CGI-Skript (falls das System nicht mit *mod_perl* oder anderen Tricks arbeitet) einen neuen Prozeß startet. Die Uhrzeit bietet auch hier eine für die Anwendung akzeptable Streuung.

Der Array *@banners* aus den Zeilen 3 bis 9 enthält eine Reihe von Referenzen auf Arrays, die als Elemente jeweils einen URL und den Namen einer Datei führen. Letztere enthält das darzustellenden Bild. Klickt der Anwender später darauf, wechselt der Browser zum zugeordneten URL.

LISTING 1

```
1  <HTML><BODY>
2  [- srand(time);
3     @banners =
4       ([„http://abc.com“, „/banners/banner1.jpg“],
5        [„http://def.com“, „/banners/banner2.jpg“],
6        [„http://ghi.com“, „/banners/banner3.jpg“],
7        [„http://jkl.com“, „/banners/banner4.jpg“],
8        [„http://mno.com“, „/banners/banner5.jpg“],
9        [„http://pqr.com“, „/banners/banner6.jpg“]);
10     $i=0;
11 -]
12 [$ while ($i++ < 3) $]
13 [- $arrayref = splice(@banners, int rand($#banners+1),
1);
14     ($href, $img) = @$arrayref;
15 -]
16       <A HREF=“[+ $href +]“>
17       <IMG SRC=“[+ $img +]“>
18       </A>
19 [$ endwhile $]
20 </BODY></HTML>
```

Aufruf über Apache

Embperl bietet keine *for*-Schleife, also initialisiert Zeile 10 zunächst die Schleifenvariable *$i*, welche anschließend die *while*-Schleife in Zeile 12 (Variablen behalten ihren Wert über Snippets hinweg bei) hochzählt, bis drei Durchläufe vollendet sind.

Der *while*-Kopf selbst steht innerhalb eines *[$ ... $]*-Tags, ist also kein Perl-Befehl, sondern ein *Embperl*-Metakommando, denn es wird nicht nur innerhalb eines Snippets, sondern über HTML-Text (Zeilen 16 - 18) und weitere Snippets hinweg iteriert. Das Ende der Schleife kennzeichnet das *[$ endwhile $]*-Metakommando in Zeile 19.

Der *splice*-Befehl in Zeile 13 holt bei jedem Schleifendurchlauf ein zufälliges Wertepaar aus dem Vorrats-Array *@banner*. Die *int*-Funktion konvertiert dabei den Fließkommawert, den die *rand*-Funktion liefert, in eine ganze Zahl. Zeile 14 wandelt die gewonnene Referenz in ein Array um und extrahiert den URL (*$href*) und den Namen der Bild-Datei (*$img*).
Die Zeilen 16 bis 18 definieren HTML-Text, der zwei Mini-Snippets enthält, die den Namen der Bilddatei und den URL in den Text einarbeiten - fertig!

Wem das *embpexec.pl* im URL zu unschön erscheint, kann auch den Apache-Server dazu bewegen, die Vorverarbeitung transparent zu übernehmen (siehe Kasten *Installation*).
Zusammen mit Doug MacEacherns *mod_perl*-Modul hilft dies *Embperl* zudem mit drastischen Performance-Verbesserungen auf die Sprünge.

Vorsicht, Beta!

Doch *Embperl* kann noch mehr: Formular-Parameter von einer Seite zur anderen durchschleifen, sowie Tabellen und Listen dynamisch generieren. Leider ist *Embperl* aber immer noch im Beta-Zustand. So kann es schon mal passieren, daß man haareraufend nach Fehlern in seinen Perl-Snippets sucht, die letztlich völlig korrekt sind - aber von *Embperl* falsch interpretiert werden. Um wenigstens eine der erweiterten Funktionen vorzustellen hierzu ein kleines Beispiel: Alle verfügbaren Banner aus Listing 1 könnte man etwa so ausgeben (siehe Listing 2):

Verfügbare Banner

URL	Banner	Image
http://abc.com	BANNER 1	/banners/banner1.gif
http://def.com	BANNER 2	/banners/banner2.gif
http://ghi.com	BANNER 3	/banners/banner3.gif
http://jkl.com	BANNER 4	/banners/banner4.gif
http://mno.com	BANNER 5	/banners/banner5.gif
http://pqr.com	BANNER 6	/banners/banner6.gif

Abb.1: Alle verfügbaren Banner in einer Tabelle

Das Ergebnis im Browser zeigt Abbildung 1. Die magische Variable *$row* steuert dabei die vogelwilde Tabellen-Logik: *Embperl* durchläuft die Tabellendefinition und erhöht *$row* für jede Tabellenzeile (respek-

LISTING 11

```
<HTML><HEAD><TITLE>Tabelle</TITLE></HEAD>
<BODY>

[- @banners =
     ([„http://abc.com“, „/banners/banner1.jpg“],
      [„http://def.com“, „/banners/banner2.jpg“],
      [„http://ghi.com“, „/banners/banner3.jpg“],
      [„http://jkl.com“, „/banners/banner4.jpg“],
      [„http://mno.com“, „/banners/banner5.jpg“],
      [„http://pqr.com“, „/banners/banner6.jpg“]);
-]

<H1>Verfügbare Banner</H1>

<TABLE BORDER=2>
  <TR>
    <TH> URL    </TH>
    <TH> Banner</TH>
    <TH> Image </TH>
  </TR>
  <TR>
    <TD> <A HREF=“[+ $banners[$row][0] +]“>
         [+ $banners[$row][0] +]</A>          </TD>
    <TD> <IMG SRC=“[+ $banners[$row][1] +]“> </TD>
    <TD> [+ $banners[$row][1] +]             </TD>
  </TR>
</TABLE>

</BODY></HTML>
```

INSTALLATION

Gerald Richters *Embperl* liegt derzeit in Version *0.20* als *HTML-Embperl-0.20.tar.gz* unter *CPAN/modules/by-module/HTML* zur Abholung bereit. Die Installation geht, wie gewohnt, mit

```
tar zxfv HTML-Embperl-0.20.tar.gz
cd HTML-Embperl-0.20
perl Makefile.PL
```

Nun scheiden sich die Wege:

Installation als CGI

Für die Installation als simples CGI-Skript ist die Frage

```
Build with support for Apache
mod_perl?(y/n) [y]
```

mit *n* zu beantworten, und

```
make install
cp embpexec.pl /usr/local/etc/httpd/cgi-bin
```

zu starten. Dies verfrachtet die notwendigen Perl-Module ins *lib*-Verzeichnis der Perl-Installation und, zusätzlich, das CGI-Skript *embpexec.pl* ins *cgi-bin*-Verzeichnis des Web-Servers. Liegt der Server nicht unter */usr/local/etc/httpd*, muß der Pfad entsprechend angepaßt werden. Ab diesem Zeitpunkt haben Requests der Form

```
http://localhost/cgi-
bin/embpexec.pl/path/file.html
```

zur Folge, daß *embpexec.pl* die Datei *htdocs/path/file.html* auf *Embperl*-Kommandos durchsucht, diese ausführt, und dem Browser dynamisch erzeugten Text zurückgibt.

Installation als Apache-Modul

Beantwortet man die Frage

```
Build with support for Apache
mod_perl?(y/n) [y]
```

hingegen mit *y*, fragt das Installationsprogramm als nächstes nach dem Pfad zu den Apache-Sourcen. Wohl dem, der seinen Apache schon mit *mod_perl* konfiguriert und die Sourcen aufgehoben hat! Andernfalls holt man sich einfach *apache_1.2.4.tar.gz* aus *http://ftp.cs.tu-berlin.de/pub/net/www/apache/dist*, sowie *mod_perl-1.05.tar.gz* aus *modules/by-module/Apache* vom nächsten CPAN-Spiegel, entpackt die beiden im gleichen Verzeichnis und installiert sie. Anschließend plaziert

```
make install
```

die notwendigen Perl-Bibliotheken. Die Konfigurationsdatei *srm.conf* im *conf*-Verzeichnis des Apache benötigt dann noch folgenden Einträge:

```
Alias /embperl /usr/local/etc/httpd/emdocs

<Location /embperl>
SetHandler perl-script
PerlHandler HTML::Embperl
Options ExecCGI
</Location>
```

Nach dem Server-Start leitet der Apache Requests der Form

```
http://localhost/embperl/file.html
```

selbständig durch die *Embperl*-Schleuse: Das *Embperl*-Dokument *file.html* muß hierzu nur im Verzeichnis */usr/local/etc/httpd/emdocs* liegen.

tive die magische Variable *$col* für jede Tabellenspalte) um 1: *0, 1, 2, ...* Gibt eines der Snippets in der Tabelle *undef* zurück, schließt *Embperl* die Tabelle ordnungsgemäß ab, sonst geht's in die nächste Runde ... *„Don't try this at home, kids ...“*

DER AUTOR

Michael Schilli arbeitet als Web-Engineer für AOL Productions, San Mateo. Er ist Autor von „Effektives Programmieren mit Perl 5“ und unter mschilli1@aol.com zu erreichen.
Seine Homepage: http://mission.base.com/mschilli.