Ein Perl-Skript verwaltet überall verfügbare Bookmarks

Lesezeichen-Zentrale

Ob im Büro, zu Hause oder mit dem Laptop im Hotelzimmer: Wer oft durch das World Wide Web surft, will seine Bookmarks immer abrufbar haben. Sie auf allen Rechnern synchron halten ist sehr umständlich. Ein CGI-Skript hilft und macht die persönliche Bookmark-Liste überall verfügbar. Michael Schilli

Das heute vorgestellte CGI-Skript nutze ich selbst seit einiger Zeit, um meine wichtigsten Bookmarks überall griffbereit zu haben. Für das Anwählen einer in der Liste gespeicherten Website genügt ein Klick auf den Eintrag »Bookmarks« in der Toolbar des Browsers. Daraufhin erscheint eine Seite wie in Abbildung 1 zu sehen. Hinter jedem Eintrag in diesem Screenshot steht eine Reihe von klickbaren Operatoren: »+« (nach oben), »-« (nach unten) und »x« (löschen). Mit ihnen lassen sich Ordner und Links in der Hierarchie herumschieben. So hält der Anwender seine Bookmark-Liste immer schön strukturiert und übersichtlich.

Kein Abtippen dank Javascript

Mit dem Webformular, das sich unter der Bookmark-Liste befindet, fügt man in vorhandenen Ordnern neue URLs hinzu. Um eine gerade besuchte Webseite neu in die Bookmarkliste aufzunehmen, will natürlich niemand URLs von Hand abtippen. Vielmehr sollen der Titel der gerade dargestellten Webseite und ihre URL einfach per Mausklick in die Bookmark-Liste wandern.

Hierzu ist ein Griff in die Javascript-Trickkiste nötig. Moderne Browser erlauben in den definierbaren Bookmark-Einträgen der Toolbar nicht nur URLs, sondern auch Javascript-Code. Klickt der Benutzer auf einen Toolbar-Eintrag, der den folgenden Code enthält, extrahiert der Browser Titel und URL der gerade dargestellten Webseite. Dann öffnet er ein neues Fenster, ruft das Bookmark-CGI-Skript auf und füllt Titel und URL automatisch ins Webformular ein:

```
javascript:void(win=window.open(➥
'http://myserver.com/cgi/bm?a='➥
location.href+'&t='+document.title))
```

Der Benutzer wählt dann nur noch einen Ordner aus der vorhandenen Liste aus oder erstellt einen neuen und klickt auf »Submit«, um den Eintrag permanent zu speichern.

Die obige Javascript-URL gelangt im Mozilla- und Netscape-Browser über den Dialog »Lesezeichen | Lesezeichen bearbeiten« in die Toolbar-Leiste (Abbildung 2). Außer diesem Toolbar-Eintrag, der den Namen »Add« trägt, sollte auch noch der eingangs erwähnte »Bookmarks«-Eintrag in die Werkzeugleiste, er verweist einfach auf das CGI-Skript. Dann ist die weltweit verfügbare Bookmark-Liste nur noch einen Mausklick entfernt. Damit Anwender den etwas länglichen Javascript-Eintrag nicht auswendig lernen oder abtippen müssen, stellt das Skript ihn praktischerweise am unteren Ende des Formulars für ein einfaches Cut & Paste dar.

Dieses System ist auf zwei Dateien verteilt: Ein Modul »Bookmarks.pm« (siehe Listing 1) implementiert die Funktionalität der Bookmark-Liste und das Skript »bm« (siehe Listing 2) kümmert sich um die Darstellung im Webbrowser und verarbeitet Benutzereingaben.

Ordnerbaum

Die Bookmark-Hierarchie legt »Bookmarks.pm« in einer Baumstruktur ab. Das Modul »Tree::DAG_Node« von Sean Burke erzeugt und manipuliert gerichtete azyklische Graphen. Es eignet sich hervorragend dafür, die in Ordnern liegenden Bookmarks zu implementieren. Sowohl Ordner als auch Bookmarks sind Knoten (Nodes) im Graphen, der an der Wurzel (Root) beginnt. Der Array »@ISA« in Zeile 11 von Listing 1 bestimmt, dass »Bookmarks« eine von »Tree::DAG_Node« abgeleitete Klasse ist. Ein Objekt vom Typ »Bookmarks« repräsentiert einfach den Wurzelknoten des

Baums, der wiederum alle Ordner als Unterknoten enthält, die ihrerseits die Bookmarks mit URL und Text als Unterknoten enthalten (siehe Abbildung 3).

Ererbter Konstruktor

»Bookmarks.pm« definiert keinen Konstruktor »new()«. Deshalb leitet Perl den Aufruf »Bookmarks->new()« einfach an »Tree::DAG_Node« weiter. Objekte vom Typ »Tree::DAG_Node« führen neben Knoten-typischen Instanzvariablen auch ein Attribut »attributes«. Hinter diesem hängt ein Hash, in den applikationsspezifische Attribute passen. Einen neuen Bookmark-Ordner erzeugt der folgende Code-Ausschnitt:

```
Bookmarks->new({
    attributes => {
        type => "folder",
        path => "Perl",
    }
});
```

Einen neuen Bookmark-Knoten erzeugt dieses simple Konstrukt:

```
Bookmarks->new({
    attributes => {
        type => "entry",
        text => $text,
        link => $link,
    }
});
```

Abbildung 3 zeigt, wie Ordner unter der Baumwurzel hängen, die jeweils ein oder mehrere Bookmarks enthalten.

Listing 1: »Bookmarks.pm«

```
001 ##########################################
002 package Bookmarks;
003 ##########################################
004 # Administer browser bookmarks
005 # Mike Schilli, 2004, m@perlmeister.com
006 ##########################################
007
008 use Storable;
009 use CGI qw(:all *dl *dt);
010 use Tree::DAG_Node;
011 our @ISA = qw(Tree::DAG_Node);
012
013 ##########################################
014 sub insert {
015 ##########################################
016     my($self, $text,
017        $link, $folder_name) = @_;
018
019     my $folder;
020
021       # Search folder node
022     for($self->daughters()) {
023       if($_->attributes()->{path} eq
024          $folder_name) {
025         $folder = $_;
026         last;
027       }
028     }
029
030       # Not found? Create it.
031     unless(defined $folder) {
032       $folder = $self->new_daughter(
033         { attributes => {
034             type => "folder",
035             path => $folder_name,
036           },
037         });
038     }
039
040       # Add it
041     return $folder->new_daughter(
042       { attributes => {
043           type => "entry",
044           text => $text,
045           link => $link,
046         },
047       });
048 }
049
050 ##########################################
051 sub folders {
052 ##########################################
053     my($self) = @_;
054
055     return map { $_->attributes()->{path} }
056                         $self->daughters();
057 }
058
059 ##########################################
060 sub as_html {
061 ##########################################
062     my($self, $nav) = @_;
063
064     my $html = start_dl();
065
066     for my $folder ($self->daughters()) {
067
068       $html .= dt(
069         b($folder->attributes()->{path}),
070         $nav->($folder->SUPER::address()));
071
072       for my $bm ($folder->daughters()) {
073         my $bma = $bm->SUPER::address();
074
075         my($link, $text) =
076           map { $bm->attributes()->{$_} }
077           qw(link text);
078
079         my $a = $bm->attributes();
080
081         $html .= dd(a({href => $link},
082                       $text), $nav->($bma));
083       }
084     }
085
086     $html .= end_dl();
087
088     return $html;
089 }
090
091 ##########################################
092 sub move_up {
093 ##########################################
094     my($self, $address) = @_;
095
096     my $node =
097           $self->SUPER::address($address);
098     if(my $left = $node->left_sister()) {
099         $node->unlink_from_mother();
100         $left->add_left_sister($node);
101     }
102 }
103
104 ##########################################
105 sub move_down {
106 ##########################################
107     my($self, $address) = @_;
108
109     my $node =
110           $self->SUPER::address($address);
111     if(my $right = $node->right_sister()) {
112         $node->unlink_from_mother();
113         $right->add_right_sister($node);
114     }
115 }
116
117 ##########################################
118 sub delete {
119 ##########################################
120     my($self, $address) = @_;
121
122     my $node =
123           $self->SUPER::address($address);
124     $node->unlink_from_mother();
125 }
126
127 ##########################################
128 sub restore {
129 ##########################################
130     my($class, $filename) = @_;
131     my $self = retrieve($filename) or
132       die "Cannot retrieve $filename ($!)";
133 }
134
135 ##########################################
136 sub save {
137 ##########################################
138     my($self, $filename) = @_;
139     store $self, $filename or
140         die "Cannot save $filename ($!)";
141 }
142
143 1;
```

Über das Attribut »type« unterscheidet die Applikation zwischen Ordnern und Bookmark-Einträgen. Beide Konstruktor-Aufrufe führt »Bookmarks.pm« nicht direkt aus, sondern ruft stattdessen die Methode »new_daughter()« aus »DAG_Node« auf, die wiederum ein »new()« der Applikationsklasse aufruft.

Die ab Zeile 14 in »Bookmarks.pm« definierte Methode »insert()« nimmt als Parameter den Text und die URL eines neuen Bookmark-Eintrags sowie den Namen des Ordners entgegen, in dem dieser liegen soll. Als erstes Argument kommt der Wurzelknoten herein, da der Aufruf über

```
$bm->insert(...)
```

erfolgt und »$bm« das Wurzelobjekt des Baums ist. Dessen Kinder, die Ordner, fördert in Zeile 22 die »daughters()«-Methode zutage. Im matriarchalischen »Tree::DAG_Node« gibt es nur Mütter mit Töchtern, Väter und Söhne hat der Autor Sean Burke wohl augenzwinkernd ausgespart.

Ist der angegebene Ordner nicht unter den vorhandenen, erstellt Zeile 32 einen neuen als Kind der Wurzel. Zeile 41 erzeugt anschließend den Bookmark-Eintrag als Kind des Ordners. Die ab Zeile 51 definierte Methode »folders()« gibt eine Liste der Namen aller Ordner zurück. Diese Liste nutzt später das CGI-Skript, um die bestehenden Ordner in einer Auswahlliste anzubieten.

Hausnummern

Um einen Knoten innerhalb des Baums zu identifizieren, besitzt »Tree::DAG_Node« die Methode »address()«, die den Weg von der Wurzel zum jeweiligen Knoten als Folge von Indizes beschreibt. Der zweite Eintrag (Index 1) des dritten Ordners (Index 2) hört beispielsweise auf den Namen »0:2:1«.

Umgekehrt kommt man von dieser Hausnummer auf das durch sie referenzierte Knotenobjekt, indem man sie irgendeinem Baumobjekt (zum Beispiel der Wurzel) als Parameter der »address()«-Methode übergibt:

```
my $node = $bm->address("0:2:1");
```

Die Hausnummer nutzt das CGI-Skript später um herauszufinden, von welchem Knoten der Benutzer den Navigationslink (rauf, runter, löschen) angeklickt hat. Die Methode »as_html()« ab Zeile 60 gibt eine HTML-Darstellung des Bookmark-Baums zurück und ruft für Ordner und Bookmark-Einträge eine als Referenz »$nav« übergebene Funktion auf. Sie bekommt die Hausnummer des jeweiligen Knotens als Argument.

So bestimmt das aufrufende Skript, wie die Navigationslinks jedes Eintrags aussehen. »as_html()« nutzt die praktischen Funktionen aus dem »CGI«-Modul, um HTML-Sequenzen zu erzeugen. Die

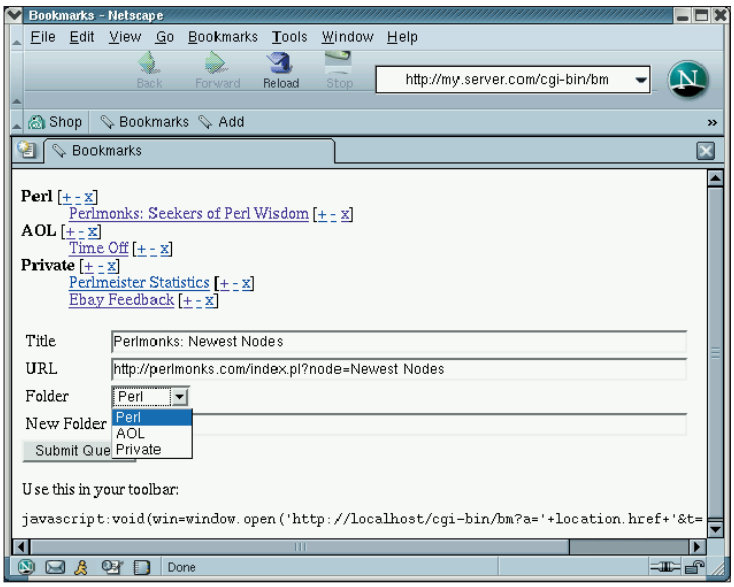

Abbildung 1: Eine weltweit verfügbare Bookmark-Liste mit Perl und CGI. Hier wird gerade ein neues Lesezeichen für die Perlmonks-Seite angelegt. Der Eintrag landet im Ordner »Perl«. Mit den Navigationselementen ist es möglich, Bookmarks zu verschieben und zu löschen.

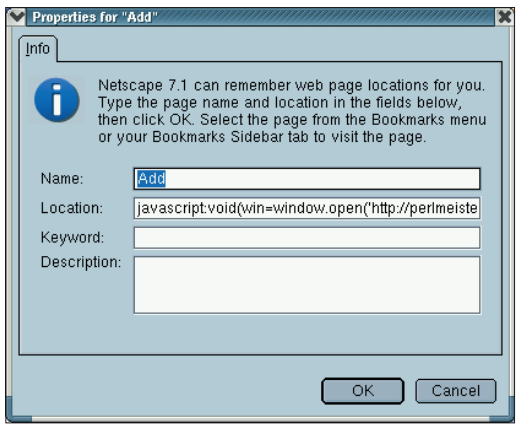

Abbildung 2: Mit Javascript lässt sich ein Toolbar-Shortcut definieren, der Titel und URL der gegenwärtig dargestellten Browserseite an das Bookmark-Skript übergibt.

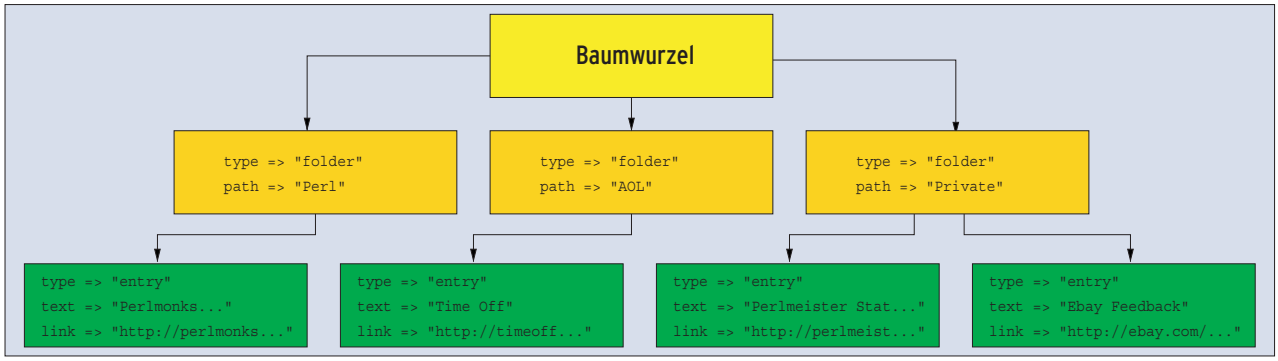

Abbildung 3: Das Perl-Modul »Tree::DAG_Node« speichert Objekte in einer Hierarchie ab. Das Bookmark-Skript nutzt diese Hierarchie, um Ordner und URLs zu speichern.

Methoden »move_up()« und »move_down()« ab Zeile 92 nehmen jeweils eine Hausnummer entgegen und befördern das damit referenzierte Knotenobjekt nach oben oder unten. Sowohl Ordner als auch Bookmark-Einträge können so innerhalb ihres Eltern-Containers umherwandern.

Das »Tree::DAG_Node«-Modul zeichnet die Kinder eines Elternknotens von links nach rechts, nicht wie in der Bookmark-Liste von oben nach unten. Die in einer Zeile aufgereihten Einträge eines Ordners starten also im Baum links mit dem ersten Eintrag und setzen sich nach rechts bis zum letzten fort.

Rauf und runter

»Tree::DAG_Node« bietet keinen direkten Weg, um einen Knoten nach links oder rechts zu verschieben. Dazu muss der Programmierer den Nachbarknoten bestimmen (über die Methoden »left_sister()« und »right_sister()«), dann den aktuellen Knoten aus dem Eltern-Container entfernen (mit »$node->unlink_from_mother()«) und ihn entweder links oder rechts vom linken oder rechten Nachbarn wieder einfügen. Genau das tun die Methoden »move_up()« und »move_down()« ab Zeile 92 mit einem über die Hausnummer referenzierten Knoten. Die »delete()«-Methode aus »Bookmarks.pm« entfernt einen Knoten vom Eltern-Container. Handelt es sich dabei um einen Ordner, verschwinden auch die in ihm enthaltenen Bookmark-Einträge.

Permanenz mit Storable

Als Datenbank, die den Zustand der Bookmark-Liste zwischen den Aufrufen des CGI-Skripts speichert, nutzt »Bookmarks.pm« das Modul »Storable«, das komplizierte und verschachtelte Datenstrukturen einfach mit »store()« speichert und mit »restore()« zurückholt. Die Methoden »save()« und »restore()« aus »Bookmarks.pm« tun dies jeweils mit dem Wurzelobjekt des Baums und schleifen so indirekt den ganzen Baum mit. Zu beachten ist, dass »store()« auf einer Instanzvariablen aufgerufen wird:

```
$bm->store($file);
```

»restore()« ist eine Klassenmethode, die einen in der angegebenen Datei abgelegten Baum ausgräbt und die Instanz eines »Bookmarks«-Objekts zurückgibt:

```
my $bm = Bookmarks->restore($file);
```

Das CGI-Skript »bm« (siehe Listing 2) übernimmt die Benutzerführung im Browser. Es zieht das Modul »CGI« für die HTML-Sequenzen herein und spezifiziert »fatalsToBrowser« für das Modul »CGI::Carp«, um im Fehlerfall schön formatierte Fehlermeldungen im Browser anzuzeigen statt sich mit einem »Internal Server Error« davonzuschleichen. Außerdem kommt das bereits erläuterte »Bookmarks«-Modul zum Einsatz.

Ab in den Browser

Das Skript setzt die Variable »$DB_FILE« in Zeile 9 auf den Namen der Datei, in der »Bookmarks.pm« den Baum permanent per »Storable::store« sichert. Zeile 20 prüft, ob Parameterwerte für URL und Text vorliegen (»a« und »t«) und ob der Submit-Knopf gedrückt wurde, was der Parameter »s« anzeigt. Das ist ein versteckter, mit dem HTML-Attribut »hidden« versehener Parameter im Webformular (ab Zeile 42).

Er sorgt dafür, dass das CGI-Skript unterscheiden kann, ob nur der Javascript-Toolbar-Eintrag Titel und URL der gerade angezeigten Webseite gesendet hat oder ob der Benutzer schon einen Ordner ausgewählt und den Submit-Knopf

Listing 2: CGI-Skript »bm«

```
01 #!/usr/bin/perl
02 ##########################################
03 # bm -- Administer bookmarks CGI
04 # Mike Schilli, 2004 (m@perlmeister.com)
05 ##########################################
06 use warnings;
07 use strict;
08
09 my $DB_FILE = "/tmp/bm.sto";
10
11 use CGI qw(:all *table);
12 use CGI::Carp qw(fatalsToBrowser);
13 use Bookmarks;
14
15 my $bm = Bookmarks->new();
16
17 $bm = Bookmarks->restore($DB_FILE) if
18                               -f $DB_FILE;
19
20 if(param('t') and param('a') and
21    param('s')) {
22   my $f = param('f');
23
24     # String overrides box selection
25   $f = param('fnew') if param('fnew');
26   die "No folder defined" unless length($f);
27
28   $bm->insert(param('t'), param('a'), $f);
29 }
30
31 $bm->delete(param('del')) if param('del');
32 $bm->move_up(param('mvu')) if param('mvu');
33 $bm->move_down(
34              param('mvd')) if param('mvd');
35
36 print header(),
37       start_html(-title => "Bookmarks");
38
39 print $bm->as_html(\&nav);
40 $bm->save($DB_FILE);
41
42 print start_form(),
43   start_table(),
44   TR(td("Title"), td(textfield(
45     -name => 't', -size => 80))),
46   TR(td("URL"), td(textfield(
47     -name => 'a', -size => 80))),
48   TR(td("Folder"), td(popup_menu(
49     -name => 'f', -values =>
50                  [$bm->folders()]))),
51   TR(td("New Folder"), td(textfield(
52     -name => 'fnew', -size => 80))),
53   end_table(),
54   hidden(s => 1),
55   submit(),
56   end_form(),
57   end_html(),
58   ;
59
60 print "Use this in your toolbar: ",
61   pre("javascript:void(win=window.open('" .
62   url(-path_info => 1) . "?a='+location." .
63   "href+'&t='+document.title))");
64
65 ##########################################
66 sub nav {
67 ##########################################
68   my($n) = @_;
69
70   return " [" .
71     a({href => url() . "?mvu=$n"},
72       "+") . " " .
73     a({href => url() . "?mvd=$n"},
74       "-") . " " .
75     a({href => url() . "?del=$n"},
76       "x") . "]";
77 }
```

gedrückt hat. Im zweiten Fall holt Zeile 22 den Namen des Ordners und Zeile 25 prüft, ob der Benutzer nicht den Namen eines neu anzulegenden Ordners (angezeigt im Parameter »fnew«) in das Textfeld eingetragen hat. Liegt kein Ordner vor, bricht Zeile 26 mit einem Fehler ab. Ansonsten fügt Zeile 28 mit der »insert()«-Methode den neuen Eintrag in die Datenbank ein.

Hat der Benutzer auf einen Navigationslink geklickt, sind entweder »del«, »mvu« (für move up) oder »mvd« (für move down) gesetzt und die Zeilen 31 bis 33 rufen die passende Methode aus »Bookmarks.pm« auf, um die Baumstruktur zu bearbeiten. Anschließend folgt die HTML-Ausgabe, eingeleitet vom HTTP-Header in Zeile 36 und gefolgt von der HTML-Repräsentation des Bookmark-Baums in Zeile 39. Zeile 40 sichert eine eventuell modifizierte Baumstruktur permanent auf Platte.

Die »print()«-Anweisung ab Zeile 42 erstellt das Webformular, das Modul »CGI« sorgt dafür, dass die Felder den vorliegenden CGI-Parametern entsprechend vorbesetzt sind. Das ab Zeile 48 erzeugte Popup-Menü mit den Namen aller existierenden Ordner entsteht mit Hilfe der in »Bookmarks.pm« definierten »folders()«-Methode. Zeile 60 gibt noch den Javascript-Eintrag aus, den der Benutzer in der Toolbar eintragen muss, damit neue Einträge einfach per Mausklick im Baum landen.

Die in Zeile 39 aufgerufene »as_html()«-Methode erhält eine Referenz auf die Funktion »nav()«, die ab Zeile 66 definiert ist. Sie gibt das HTML für die hinter jedem Eintrag angezeigte Navigationsliste zurück. Wie bereits ausgeführt ruft »as_html()« die Funktion »nav()« für jeden angezeigten Eintrag auf und übergibt die Hausnummer. Sie ist dort als »$n« verfügbar und wird an die Links angehängt, die auf das CGI-Skript selbst verweisen und Navigationsanweisungen wie »mvu«, »mvd« oder »del« enthalten.

Installation

Das Modul »Bookmarks.pm« nutzt »Tree::DAG_Node« und »Storable« vom CPAN. Sind sie installiert, muss »bm« ausführbar im »cgi-bin«-Verzeichnis des Webservers landen, das Modul »Bookmarks.pm« entweder in demselben Verzeichnis oder an einer Stelle, an der »bm« danach sucht.

Damit nicht die ganze Welt die Bookmarkliste manipulieren kann, schützt der Webserver sie mit einer ».htaccess«-Datei:

```
AuthType Basic
AuthName "Mike's Bookmarks"
AuthUserFile /var/www/htpasswd
Require valid-user
```

Diese Methode ist zwar nicht besonders sicher, da der Browser im Basic-Auth-Verfahren das Passwort quasi im Klartext übers Netz sendet, aber sie bietet immerhin einen rudimentären Schutz.

Einschränkungen

Das Skript geht davon aus, dass jeweils nur ein Anwender es nutzt, und trifft keine Vorkehrungen, um Zugriffe auf die permanenten Daten zu synchronisieren. Es handelt außerdem nach der Unix-Philosophie, dass ein fortgeschrittener Benutzer immer wissen sollte, was er tut: Einmal auf das »x« eines Ordners geklickt – und schon verschwindet dieser mitsamt der in ihm enthaltenen Links ohne Nachfrage im Nirvana.

Wer sich für die in der Datenbankdatei abgelegte Datenstruktur interessiert, erzeugt mit Hilfe des »dumpsto«-Skripts [2] einen Dump der Storable-Datei. Anschließend befördert »dumpsto -u« die Daten wieder in ein File. *(mwe)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://www.linux-magazin.de/pub/listings/magazin/2004/05/Perl]

[2] Dumpsto und andere Skripte in „Mike's Script Archive": [http://perlmeister.com/scripts]

Der Autor

Michael Schilli arbeitet als Software-Engineer für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5" (deutsch) und „Perl Power" (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage ist [http://perlmeister.com].