Using a Perl script to manage globally accessible book#

Global Memory

While surfing the Web from your office, your home, or even with a laptop in a hotel, a globally accessible CGI script ensures that your personal bookmark list will be available any time, anywhere. BY MICHAEL SCHILLI

Have you ever bookmarked a site in your browser and not been able to find it later because you've switched to a different machine? The CGI script that we will be looking at has been in use on my website for a month now and I can't live without it. A single click on a brand new "Bookmarks" entry in the toolbar pointing to the script is all it takes to pull up the globally available bookmark list like in Figure 1.

The script displays bookmarked sites as links, and a number of operators next to each entry allows you to move them: + (up), - (down), and *x* (delete). Bookmarks are organized in folders.

Javascript Saves Typing

The HTML form below the list accepts URLs, text and folder names for new bookmark entries. Nobody wants to type in a lengthy URL just to add a site.

A little Javascript magic is all this takes. Modern browsers allow Javascript code beside URLs as editable bookmark entries in the toolbar. When a user clicks on a toolbar entry that contains the following code, the browser will extract the title and URL from the website just viewed in the browser, call the bookmark CGI script, and enter the title of the page and its URL into the Web form:

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

All the user needs to do, is select a folder from the list, or enter the name of a new folder, and click on *Submit*.

To add the *Bookmarklet* to the Mozilla / Netscape toolbar, select *Bookmarks | Manage Bookmarks*. See Figure 2. Besides this toolbar entry, labeled "Add", you will also need a "Bookmarks" entry pointing to the script, if you just want to pull up the bookmark list. Then you are just one click away from a globally accessible bookmark list. The Javascript entry is fairly complex, but you do not need to memorize it. The script displays the entry at the bottom of the page, and cut & paste takes care of the rest.

The Perl code for the bookmark system spans two files: the *Bookmarks.pm* module (see Listing 1) which implements the functionality of the bookmark list itself, and the *bm* CGI script (see Listing 2) which takes care of user input and generates HTML output for the browser.

Directory Tree

Bookmarks.pm creates a tree structure. Sean Burke's *Tree::DAG_Node* module creates and manipulates directed, acyclic graphs and is really well-suited to implementing the bookmarks within the folder structure. Both folders and bookmarks are nodes in a graph that has its origin at a root node. The *@ISA* array in line 12 of Listing 1 contains *Tree::DAG_Node*, making *Bookmarks* a subclass *Tree::DAG_Node*. An object of the *Bookmarks* class represents the root node of the tree. The root node has child nodes for all folders, which in turn have child nodes for the bookmarks with their URLs and text content (see Figure 3).

Inherited Constructor

Bookmarks.pm does not define a *new()* constructor. Instead, *Bookmarks->new()* is simply passed on to the superclass *Tree::DAG_Node*. *Tree::DAG_Node* type objects have an attribute called *attributes* beside their typical node instance variables. *attributes* points to a hash that can store application specific attributes. The following code snippet creates a new bookmark folder:

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

The following construct creates a new node with a bookmark *entry*:

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

Figure 3 shows a number of folders below the root of the tree structure; each containing one or multiple bookmarks.

The module uses the *type* attribute to distinguish between folder and bookmark entries. *Bookmarks.pm* does not issue the constructor calls directly. It uses the *new_daughter()* method of the superclass, which calls *new()* of the application class under the hood.

The *insert()* method defined in lines 15 ff. in *Bookmarks.pm* expects the title and URL for a new bookmark entry, and the name of the folder in which to store the entry, as parameters. Because it is a method call, the first argument is the root node:

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

Its "daughter" nodes, that is the folders, are retrieved by the *daughters()* method

in line 23. *Tree::DAG_Node* creates a matriarchy, as its author, Sean Burke, refers to “mothers” and “daughters”.

If the folder does not exist, line 34 will create it as a child of the root node. Line 43 creates a bookmark entry as a child of the folder node.

The *folders()* method defined in lines 53 ff. returns a list of folder names. The CGI script will use this list later to create a pull-down list of available folders.

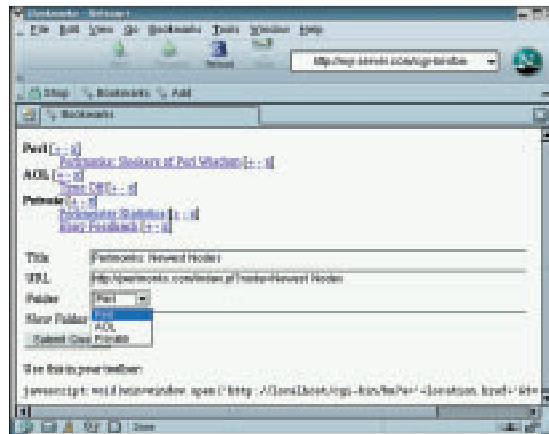

Figure 1: The globally accessible bookmark list. Creating a new bookmark for the Perlmonks page.

House Numbers

Every *Tree::DAG_Node* object has an *address()* method to identify nodes within a tree. The method describes the path from the root to the current node as a sequence of indices. The second entry (index 1) of the third folder (index 2) is thus referred to as *“0:2:1”*.

This “house number” can be used to access the node object, by passing it to any tree object (for example the root) as a parameter of the *address()* method:

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

The CGI script will use the house number to discover the node within which the user has clicked a navigational link (up, down, delete). The *as_html()* method in line 63 ff. of *Bookmarks.pm* returns a HTML rendering of the bookmark tree, and calls a function, referred to by *$nav*, for each of the folders and bookmark entries. The function expects the house number of the node as its argument and spits out the HTML for the navigational links. *as_html()* makes use of the handy functions in *Lincoln Stein*‘s *CGI* module to create HTML sequences.

Up and Down

The *move_up()* and *move_down()* methods in line 102 ff. and 120 ff. both expect a house number, and will move the node object up and down accordingly.

The *Tree::DAG_Node* module imagines the child nodes of parent nodes from left to right. Thus, the first entry within a folder is the left-most child of the parent node. *Tree::DAG_Node* does not provide a direct method for moving a node left or right. To do this, we’ll dig up the neighboring nodes, using *left_sister()*, and *right_sister()*), remove the current node from the parent container (*$node->unlink_from_mother()*), and then insert it either left or right of the neighbor. The *delete()* method in line 138 ff. removes a node from the parent container. If the node is a folder, any bookmark entries are also deleted.

Permanence Using Storable

To provide a database that stores the state of the bookmark list between CGI script calls, *Bookmarks.pm* uses the *Storable* module, which stores complicated, nested data structures by simple calls to *store()*, using *restore()* to reinstate them. The *save()* and *restore()* methods in *Bookmarks.pm* both use the root object of the tree to do this, thus dragging along the rest of the tree. Note that *store()* refers to an instance variable, as in

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

whereas *restore()* is a class method

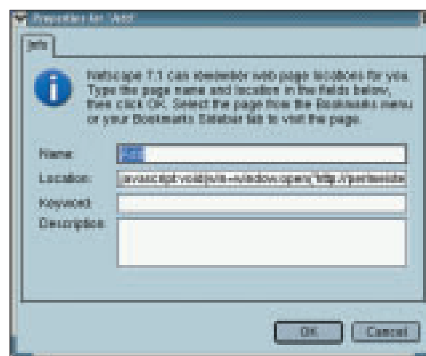

Figure 2: Using Javascript to define a toolbar shortcut.

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

to extract a tree from a file, and assign it to the instance of a *Bookmarks* object.

CGI into the Browser

The CGI script, *bm*, handles user management. It also uses the *CGI* module for the HTML sequences and specifies *fatalsToBrowser* for *CGI::Carp*, to provide nicely formatted output in the browser if something goes wrong, instead of the dreaded *Internal Server Error*. It also pulls in the *Bookmarks.pm* module.

$DB_FILE in line 10 contains the name of the file in which *Bookmarks.pm* will store the tree permanently, using *Storable::store*. Line 22 checks if the values for the URL and text parameters (*u* and *t*) exist, and if the Submit button has been pressed. The *s* parameter, a *hidden* HTML parameter in the Web form takes care of this (line 71 ff.). This lets the CGI script determine whether the Javascript toolbar entry has just sent the title and URL for the website entry, or if the user added a new entry.

In the latter case, line 25 retrieves the folder name, and line 29 checks if the user has entered a non-existent folder name (shown in the *fnew* parameter) into the textbox. If the folder name is missing, line 30 quits with an error message. Otherwise, line 33 will call the *insert()* method to add to the database.

When a user clicks a navigational link, either *del*, *mvu* (for “move up”), or *mvd* (for “ > move down”) is set, and lines 37 through 39 call the appropriate *Bookmarks.pm* method to modify the tree structure. The HTML output follows, starting with the HTTP header in line 44, and followed by the HTML rendering of the bookmark tree in line 48. Line 49 puts the modified tree structure into permanent storage on the disk.

The *print()* command in line 51 creates the web form. The *CGI* module automatically enters CGI parameter values into the form fields. The pop-up menu with the names of the existing folders created in line 61 ff. uses the *folders()* method in *Bookmarks.pm*.

Line 76 outputs the bookmarklet mentioned earlier. After adding this entry to the toolbar, users can point and click to create new entries in the tree structure.

Listing 1: Bookmarks.pm

```
001 ##############################
002 package Bookmarks;
003 ##############################
004 # Admin browser bookmarks
005 # Mike Schilli, 2004
006 # m@perlmeister.com
007 ##############################
008
009 use Storable;
010 use CGI qw(:all *dl *dt);
011 use Tree::DAG_Node;
012 our @ISA= qw(Tree::DAG_Node);
013
014 ##############################
015 sub insert {
016 ##############################
017   my($self, $text, $link,
018      $fname) = @_;
019
020   my $folder;
021
022     # Search folder node
023   for($self->daughters()) {
024     if($_->
025       attributes()->{path} eq
026       $fname) {
027         $folder = $_;
028         last;
029     }
030   }
031     # Not found? Create it.
032   unless(defined $folder) {
033     $folder =
034       $self->new_daughter({
035         attributes => {
036           type => "folder",
037           path => $fname,
038         },
039       });
040   }
041       # Add it and return obj
042   return $folder->
043     new_daughter({
044       attributes => {
045         type => "entry",
046         text => $text,
047         link => $link,
048       },
049     });
050 }
051
052 ##############################
053 sub folders {
054 ##############################
055   my($self) = @_;
056
057   return map {
058     $_->attributes()->{path}
059   } $self->daughters();
060 }
061
062 ##############################
063 sub as_html {
064 ##############################
065   my($self, $nav) = @_;
066
067   my $html = start_dl();
068
069   for my $f ($self->
070                daughters()) {
071
072     $html  .= dt(
073       b($f->attributes()->
074           {path}),
075       $nav->(
076         $f->SUPER::address())
077     );
078
079     for my $bm ($f->
080                daughters()) {
081       my $bma =
082        $bm->SUPER::address();
083
084       my($link, $text) =
085         map { $bm->
086          attributes()->{$_}
087         } qw(link text);
088
089       $html  .= dd(
090         a({href => $link},
091           $text
092          ), $nav->($bma));
093     }
094   }
095
096   $html  .= end_dl();
097
098   return $html;
099 }
100
101 ##############################
102 sub move_up {
103 ##############################
104   my($self, $address) = @_;
105
106   my $node =
107     $self->SUPER::address(
108                $address);
109
110   if(my $left =
111       $node->left_sister()) {
112     $node->
113         unlink_from_mother();
114     $left->
115         add_left_sister($node);
116     }
117 }
118
119 ##############################
120 sub move_down {
121 ##############################
122     my($self, $address) = @_;
123
124     my $node =
125        $self->SUPER::address(
126                   $address);
127     if(my $right =
128      $node->right_sister()) {
129       $node->
130         unlink_from_mother();
131       $right->
132         add_right_sister(
133                  $node);
134     }
135 }
136
137 ##############################
138 sub delete {
139 ##############################
140   my($self, $address) = @_;
141
142   my $node =
143        $self->SUPER::address(
144                   $address);
145   $node->
146         unlink_from_mother();
147 }
148
149 ##############################
150 sub restore {
151 ##############################
152   my($class, $filename) = @_;
153
154   my $self =
155     retrieve($filename) or
156     die "Cannot retrieve " .
157         "$filename ($!)";
158 }
159
160 ##############################
161 sub save {
162 ##############################
163   my($self, $filename) = @_;
164
165   store $self, $filename or
166     die "Cannot save " .
167         "$filename ($!)";
168 }
169
170 1;
```

Listing 2: bm

```
01 #!/usr/bin/perl
02 ###############################
03 # bm -- Global Bookmarks CGI
04 # Mike Schilli, 2004
05 # (m@perlmeister.com)
06 ###############################
07 use warnings;
08 use strict;
09
10 my $DB_FILE = "/tmp/bm.sto";
11
12 use CGI qw(:all *table);
13 use CGI::Carp qw(
14             fatalsToBrowser);
15 use Bookmarks;
16
17 my $bm = Bookmarks->new();
18
19 $bm = Bookmarks->restore(
20     $DB_FILE) if -f $DB_FILE;
21
22 if(param('t') and param('a')
23    and param('s')) {
24
25   my $f = param('f');
26
27     # String overrides select
28   $f = param('fnew')
29             if param('fnew');
30   die "No folder defined"
31            unless length($f);
32
33   $bm->insert(param('t'),
34               param('a'), $f);
35 }
36
37 $bm->delete(param('del')) if
38                 param('del');
39 $bm->move_up(param('mvu')) if
40                 param('mvu');
41 $bm->move_down(param('mvd'))
42               if param('mvd');
43
44 print header(),
45       start_html(
46       -title => "Bookmarks");
47
48 print $bm->as_html(\&nav);
49 $bm->save($DB_FILE);
50
51 print start_form(),
52   start_table(),
53   TR(td("Title"),
54      td(textfield(
55           -name => 't',
56           -size => 80))),
57   TR(td("URL"), td(textfield(
58           -name => 'a',
59           -size => 80))),
60   TR(td("Folder"),
61      td(popup_menu(
62           -name => 'f',
63           -values =>
64             [$bm->folders()]
65          ))),
66   TR(td("New Folder"),
67      td(textfield(
68           -name => 'fnew',
69           -size => 80))),
70   end_table(),
71   hidden(s => 1),
72   submit(),
73   end_form(), end_html(),
74   ;
75
76 print "Use this in your " .
77       "toolbar: ", pre(
78   "javascript:void(win=" .
79   "window.open('" .
80     url(-path_info => 1) .
81   "?a='+location.href+'&t='".
82   "+document.title))");
83
84 ###############################
85 sub nav {
86 ###############################
87   my($n) = @_;
88
89   return " [" .
90     a({href => url() .
91       "?mvu=$n"}, "+") . " " .
92     a({href => url() .
93       "?mvd=$n"}, "-") . " " .
94     a({href => url() .
95       "?del=$n"}, "x") . "]";
96 }
```

The *as_html()* method called in line 48 gets a reference to the *nav()* function, which is defined in lines 85 ff.

The function is responsible for returning the HTML code for the navigational elements for each of the entries. As previously explained, *as_html()* calls *nav()* for each of the entries displayed, and passes the house number for that entry. The number gets assigned to *$n* in *nav()*, and it is appended to links that refer to the CGI script and contain navigational commands such as *mvu*, *mvd*, and *del*.

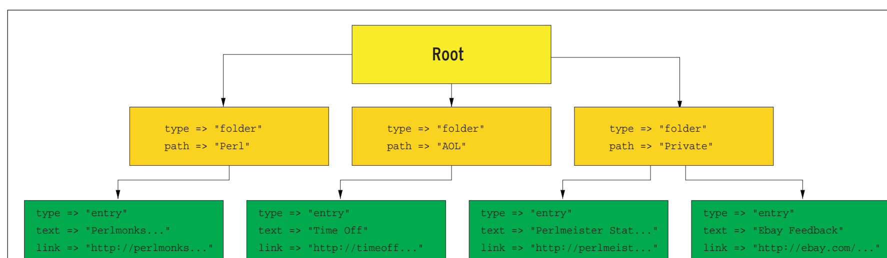

Figure 3: The *Tree::DAG_Node* Perl module stores objects in a tree structure. The bookmark script leverages this structure to store folders and URLs.

Installation

The *Bookmarks.pm* module requires *Tree::DAG_Node*, and *Storable* from CPAN. Copy *bm* to the *cgi-bin* directory of the Web server and make it executable. Copy *Bookmarks.pm* to the same directory, or where *bm* can find it.

To protect your bookmark list, your Web server can use a *.htaccess* file:

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

The *Basic Auth* scheme uses a password for every permitted user, which can be set with

```
htpasswd username ⮐
/var/www/htpasswd
```

Basic Auth isn't the safest method, but it's fine for my purposes.

Restrictions

The script assumes only one user.

If you are interested in the data structure stored in the data file, you can use the *dumpsto* script [2] to create a dump of the storable file. *dumpsto -u* will put the data back in a file.

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/43/Perl/*

[2] *dumpsto* and other scripts in "Mike's Script Archive": *http://perlmeister.com/scripts*