

A flexible multi-level UL/LI constructor

Source: <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2007-09/msg00348.html>

- *From:* Tuxedo <tuxedo@xxxxxxxxxxxxxxxx>
 - *Date:* Sat, 08 Sep 2007 22:27:09 -0400
-

Hi!

Some time ago I received much help on this group in piecing together a fairly simple two-level HTML list but where conditional output depending on a current page variable and various other peculiarities were required.

For a similar but somewhat more complex purpose, I'd like find a suitable method of generating the following nested HTML list structure:

- o Heading 1
 - o Subject 1.1
 - o Subject 1.2
 - o Sub heading 1.3
 - o Subject 1.3.1
 - o Subject 1.3.2
 - o Subject 1.4
 - o Subject 1.5

Ultimately, the list will be converted by CSS and Javascript into a typical drop-menu. This allows text browsers to gracefully fall back on the original list structure, while presenting most people with a fancy drop-menu. The complete HTML output would appear as follows:

```
<div class="menu">

<ul>

<li class="expand"><a href="heading1.html">Heading 1</a>

<ul>

<li><a href="subject1.1.html">Subject 1.1</a></li>
<li><a href="subject1.2.html">Subject 1.2</a></li>

<li class="expand"><a href="subheading1.3.html">Sub heading 1.3</a>

<ul>
<li><a href="subject1.3.1.html">Subject 1.3.1</a></li>
<li><a href="subject1.3.2.html">Subject 1.3.2</a></li>
```

A flexible multi-level UL/LI constructor

```
</ul>

</li>

<li><a href="subject1.4.html">Subject 1.4</a></li>
<li><a href="subject1.5.html">Subject 1.5</a></li>

</ul>

</li>

</ul>

</div>
```

The above example would generate one multi-level drop-menu only, while in reality, the complete navigation system will consist of several drop-menus which are all aligned horizontally in a typical page header style.

Some menus may include two, three, four and even five level UL's. One level only can also exist in the odd situation, which would simply display a top-level link without a drop-menu.

Each menu will be placed within separate `<div class=menu>` tags. In short, without href's, classes and ID's, an example of two menus follows below. In this case the first menu has three levels and the second has four levels:

```
<!-- first menu, containing UL's and LI's with maximum three levels -->

<div class="menu">

<ul>
<li>Heading 1

<ul>

<li>Subject 1.1</li>
<li>Subject 1.2</li>

<li>Sub heading 1.3

<ul>
<li>Subject 1.3.1</li>
<li>Subject 1.3.2</li>
</ul>

</li>

<li>Subject 1.4</li>
<li>Subject 1.5</li>
```

A flexible multi-level UL/LI constructor

```
</ul>

</li>

</ul>

</div>

<!-- second menu, containing UL's and LI's with maximum four levels -->

<div class="menu">

<ul>

<li>Heading 2

<ul>

<li>Subject 2.1</li>

<li>Sub heading 2.2

<ul>

<li>Subject 2.2.1</li>
<li>Sub heading 2.2.2

<ul>

<li>Subject 2.2.2.1</li>
<li>Subject 2.2.2.2</li>

</ul>

</li>

</ul>

</li>

</ul>

</li>

<li>Subject 2.3</li>
<li>Subject 2.4</li>
<li>Subject 2.5</li>
<li>Subject 2.6</li>
<li>Subject 2.7</li>

</ul>

</li>

</ul>

</div>
```

A flexible multi-level UL/LI constructor

The above first-level UL's/LI's ("Subject 1" and "Subject 2"), always contain only one link in it's own level, followed directly by the opening of a second-level nested UL. This allows CSS to style all others initially as hidden, until the first level heading is hovered. But this posting is not about CSS and so I have excluded any such code. However, should anyone want the accompanying CSS, I would be happy to post it here. In fact, different types of CSS menus can be built upon the same or similar type of UL/LI structures and these can naturally only work as expected if the correct list structures are in place to start with.

The above examples shows the basic UL/LI nesting which the Perl script needs to generate. However, some additional details needs to be considered to know exactly what type of coding methods may best serve the particular purpose.

For example, the Perl script needs to recognize which LI's are followed by a nested UL and print such entries with a specific HTML class:

```
<li class="expand">
```

This allows CSS to place a visual indicator, such as, a small arrow (>) next to the linked area from where sub-menus unfold when hovered. For example, when hovering Sub heading 1.3, it's nested sub-menu, including Subject 1.3.1 and 1.3.2, appear directly to the right of the hovered area:

```
+-----+
| Heading 1 > | Heading 2 > | Heading 3 > | etc.
+-----+
| Subject 1.1 |
+-----+
| Subject 1.2 |
+-----+
| Sub heading 1.3 > | Subject 1.3.1 |
+-----+
| Subject 1.4 | Subject 1.3.2 |
+-----+
| Subject 1.5 |
+-----+
```

The Perl program will also need to know where within a menu the current HTML page is represented by traversing the various href values in arrays or hashes. The current page is identified by a CGI environment variable as:

```
$current::page = $ENV{DOCUMENT_NAME} || 'undefined';
```

All HTML documents will exist in one directory level (DocumentRoot) and no page will be linked to twice from the navigation menus. Therefore, a same DOCUMENT_NAME conflict will never occur in any of the Perl references.

The LI entry matching the current document should appear as a non-link and with a unique HTML ID. For example, if the \$current::page matches subject1.3.1.html, then that entry should be printed in the menu as:

A flexible multi-level UL/LI constructor

```
<li id=CURRENT_PAGE>Subject 1.3.1</li>
```

Furthermore, the program will need to be aware of the navigational pathway, via the arrays or hashes, so it can:

1) Return a text string that displays where the user is, so if the current page is subject1.3.1.html, then this string would become:

You are in: Heading 1 > Sub heading 1.3 > Subject 1.3.1

2) Output a specific HTML class in the relevant href tags in the pathway, so when the current page is subject1.3.1.html, a class, e.g. "PATH", would apply to the entries at the nesting positions that lead up to the current page. In this case, these two separate HTML tags would appear as:

```
<a href="heading1.html" class="PATH">Heading 1</a>
```

```
<a href="subheading1.3.html" class="PATH">Sub heading 1.3</a>
```

What signifies pathway LI's is that they can only exist directly after the opening of LI's with opening UL's before the corresponding LI is closed. Heading 1 and Sub heading 1.3 in this case are both nesting points and the pathway to the current page.

Apart from the existence of the menu header, subheading1.3.html will not present any special content. These type of pages only summarize content that exist on the sub-menu pages which they represent. Such pathway pages, including those at first levels, i.e. heading1.html and heading2.html, fill the same type of purpose in this navigation system, i.e. they are simply Tables of Contents. Anyhow, this has more to do with the site's content organization than with the actual functions of the Perl program itself.

Would some kind of multi-level hash arrays, perhaps using Tie::IXHash, be an appropriate method to generate these type of menu lists?

The procedure is only meant to output dynamically generated menus across all pages of a site through SSI within a localhost environment. In other words, the script will not run for each and every page request in a real web server situation, so no system performance limitations apply. A different procedure will save all pages in static form to transfer onto a web server.

The projects objective is to maintain a navigation menu across a site via one Perl script, allowing for navigation links of each separate menu to be easily modified across the site, including the varying levels of nesting.

In summary, the Perl program should ideally do the all the following:

- 1) Generate the multi-level UL/LI structure defined in hashes or arrays.
- 2) Print the current page menu entry differently from others.
- 3) Identify and print LI's of nesting points differently, as well as return

A flexible multi-level UL/LI constructor

the navigational pathway string.

What would be a good way to glue these points together? How can the first point be constructed with a suitable loop for example?

Any ideas, including barebone code examples, that can help lead me in the right direction, would be greatly appreciated!

Many thanks,
Tuxedo

—

One thing the inventors can't seem to get the bugs out of is fresh paint.

.