

HTML::App

A toolkit for Perl (mason) Templating

Jim Krajewski

Current State of HTML Templating in Perl

- Main component files (.html)
- Subcomponent files

Main component files

- Templating flexible and works well.

...

Main component files

- Templating flexible and works well.

```
% if( defined($message) ) {  
    <div class="status">  
        <span class="message">  
            <% $message %>  
        </span>  
    </div>  
  
    <br>  
% }
```

Main component files

- The problem is, we build out HTML elements inside the template (e.g.)

...

Main component files

- The problem is, we build out HTML elements inside the template (e.g.)

```
<select name="groupSelect" id="groupSelect" size="6">
%   my $last_group_name = "";
%   while( my $group_row = $sth->fetch() ) {
%       if( $last_group_name ne $group_row->{group_name} ) {
%           $last_group_name = $group_row->{group_name};
%           <option value="<% $group_row->{groupid} %"><% $group_row->{group_name} %></option>
%       }
%   }
%   </select>
```

Subcomponent files

- They're reusable

Subcomponent files

- They're reusable

```
<%args>
  $id
  $name
  $label
  $size => 6
  @opt_list
</%args>
<select name="<% $name %>" id="<% $id %>" size="<% $size %>">
%   foreach my $opt (@opt_list) {
%       <option name="<% $opt->{value} %>"><% $opt->{label} %></option>
%   }
</select>
```

Subcomponent files

- They're reusable

Subcomponent files

- They're reusable
- They can take Perl structured values (references).

Subcomponent files

- Problems:
 - Forms – you can't always embed `<form>` tags in a component.
 - Namespace – **id** and **name** attributes are scoped to the document and not a component. Even if you use a naming convention, you can't easily put more than one on a form.

Issues for making
components reusable

...

Issues for making components reusable

- HTML **id** and **name** attributes being in a flat namespace are essentially global variables.

Issues for making components reusable

- HTML **id** and **name** attributes being in a flat namespace are essentially global variables.
- Although subcomponents can take an Perl references (thus structured values), main components called via GET or POST can only take strings.

Issues for making components reusable

- HTML **id** and **name** attributes being in a flat namespace are essentially global variables.
- Although subcomponents can take an Perl references (thus structured values), main components called via GET or POST can only take strings.
- HTML rendering can be encapsulated in subcomponent files, but the form actions are sent by the web browser at the page level, breaking encapsulation.

My Solution:

HTML::App

HTML::App features:

- No need to assign global **id** or **name** attributes.

The **key** property is used instead. It is relative to its place in the widget hierarchy similar to names in a file system. A widget is uniquely identified by its **key path**.

id and **name** attributes are autogenerated as needed but can be specified if desired.

HTML::App features:

- Values can be encoded as a JSON string.
 - Asking a composite widget for its value will return a composite value. Values can nest to any level.
 - JSON-encoded values sent to the web server in query or post operations are decoded back into the corresponding Perl structure.

HTML::App features:

- You can more easily create reusable subcomponents:
 - To handle the rendering of HTML, send into the subcomponent a data structure or the values (to whatever nesting level necessary).
 - To parse the form data, simply ask the subcomponent for its JSON value. If it's a composite subcomponent, the value returned is the composite of all its children's values as a single string.

demo.html

How HTML::App is organized:

- Two primary base classes:
 - HTML::App::TextNode
 - HTML::App::ElemNode

What they do:

- They both have a value and can render themselves as HTML text.
- They can both be members of a Node Hierarchy.

HTML::App::TextNode

- Abstracts an HTML DOM text node.
- Can't have child nodes – only a leaf node.
- Can render itself as inline text, but first passes itself through `maketext()` – it transparently handles language translation.

HTML::App::ElemNode

- Abstracts an HTML DOM element node. It holds the standard HTML attribute and class properties of an HTML element as well as its own properties.
- Can have child nodes to form a hierarchy.
- Can render itself as HTML text. If it has children, the whole hierarchy is recursively rendered.
- Can be subclassed.
- By subclassing and adding a corresponding JavaScript class, it becomes a *widget*.

Widgets

- Reusable HTML::App components (subclasses of HTML::App::ElemNode).
- Have a corresponding JavaScript Widget class.

JavaScript Widget Class

- Is a JavaScript class – it has properties and methods and can be subclassed.
- Represents and operates on an HTML hierarchy.
- Provides consistent, normalized accessors: `getValue()`, `setValue()`, `jsonValue()`.
- Can have event handlers.

Working with JavaScript Widget Classes

- Web developers work with the JS object – not with the HTML elements[†]. Event handlers reference the object (*this*)
- Widgets are assigned a *key*, and are referenced by a *key path*, either absolute or relative.

[†] However, there still is direct access to the HTML elements, if necessary.

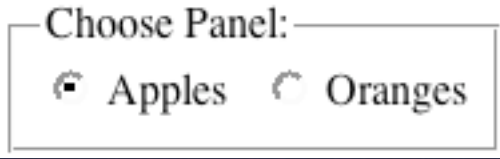
Types of Widgets

- **Simple Widgets:** (usually composed of one HTML element)
 - Buttons
 - Checkboxes
 - Radio Buttons[†]
 - Text, Text Edit, Textarea

[†] Never used alone.

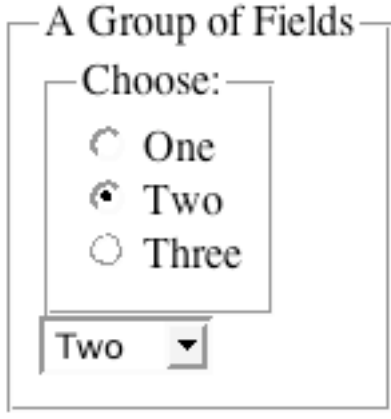
Types of Widgets

- Intermediate Widgets:
 - Selects
 - Radio Groups
 - Checkbox Groups?
 - Field Groups



Choose Panel:

Apples Oranges



A Group of Fields

Choose:

One
 Two
 Three

Two ▼

Types of Widgets

- Complex Widgets:
 - Tables
 - Tab Groups
 - Panel Views
 - Calendar?

| Pri- ority | Name | Address | City | State | Zip Code |
|---------------|-------|---------------|--------------|-----------|----------|
| 1 | Jim | Front Street | Lisle | Illinois | 60532 |
| 2 | Joe | Center Street | Lisle | Illinois | 60532 |
| 3 | Jill | Back Street | Indianapolis | Indiana | 45532 |
| 4 | John | Frontage Road | LaCross | Wisconsin | 34232 |
| 5 | Jenny | Back Street | Minneapolis | Minnesota | 98463 |
| | name | | | | zip |

| | | |
|--|-------|----|
| Man | Mouse | Me |
| Walt Disney 1313 S Harbor Blvd Anaheim, CA 92802 847/555-1212 | | |

What's Next?

- Where can I find it?
- How do I use it?
- Sample code.

Where can I find it?

- <http://www.echosoft.com/html/downloads.html>
- <http://www.JSON.org/json2.js>

How do I use it?

- Put the following in the `<head>` section:

```
<link rel='stylesheet' type='text/css' href="/styles/HtmlApp.css" />
```

```
<script type="text/javascript" language="Javascript1.2" src="/scripts/json2.js"></script>
```

```
<script type="text/javascript" language="Javascript1.2" src="/scripts/HtmlApp.js"></script>
```

```
<link rel='stylesheet' type='text/css' href="/styles/HATable.css" />
```

```
<script type="text/javascript" language="Javascript1.2" src="/scripts/HATable.js"></script>
```

- Sample code fragments can be found in the `examples/` directory of the package file.

Questions