Template::Flute - Modern HTML and PDF Engine

Stefan Hornburg (Racke)
racke@linuxia.de

Perl Mongers Hamburg, 4th April 2011

Contents

1 Template::Flute

Template::Flute enables you to completely separate web design and programming tasks for dynamic web applications.

Templates are plain HTML files without inline code or mini language, thus making it easy to maintain them for web designers and to preview them with a browser.

The CSS selectors in the template are tied to your data structures or objects by a specification, which relieves the programmer from changing his code for mere changes of class names.

In addition to HTML output, Template::Flute also supports generation of PDF files on-the-fly based on the same template and specification.

1.1 Why and Where

Why

- Separation of web design and programming
- How available template engines violate this principle
  - Mini language (Template::Toolkit)
  - Inline code
  - CSS selectors (HTML::Zoom)
- Solutions by Template::Flute
  - Static HTML file
  - Specification file
- Further Goals

  - Great flexibility
  - Tweaks through tree manipulations

1.2 Cart Example

1.2.1 Cart as Hash

Cart: Hash

```
$cart = [
         {isbn => '978-0-2016-1622-4',
          title => 'The Pragmatic Programmer',
          quantity => 1, price => 49.95},

         {isbn => '978-1-4302-1833-3',
          title => 'Pro Git',
          quantity => 1, price => 34.99},
      ];
```

1.2.2 HTML Template

Cart: HTML Template

```
<table class="cart">
<tr class="cartheader">
<th>Name</th><th>Quantity</th><th>Price</th>
</tr>
<tr class="cartitem">
<td class="name">Perl 6</td>
<td><input class="quantity" name="quantity" size="3" value="10"></td>
<td class="price">$$$</td>
</tr>
<tr class="cartheader">
<th colspan="2"></th><th>Total</th></tr>
<tr>
<td colspan="2"></td><td class="cost">$$$</td></tr>
</table>
```

1.2.3 Cart with ITL

Cart: ITL

```
<table class="cart">
<tr class="cartheader">
<th>Name</th><th>Quantity</th><th>Price</th>
</tr>
[item-list]
<tr class="cartitem">
<td class="name">[item-modifier title]</td>
<td><input class="quantity" name="quantity" size="3" value="[item-quantity]"></td>
<td class="price">[item-price]</td>
</tr>
[/item-list]
<tr class="cartheader">
```

```
<th colspan="2"></th><th>Total</th></tr>
<tr>
<td colspan="2"></td><td class="cost">[total-cost]</td>
</tr>
</table>
```

1.2.4 Cart with Template::Toolkit

Cart: Template::Toolkit

```
<table class="cart">
<tr class="cartheader">
<th>Name</th>
<th>Quantity</th>
<th>Price</th>
</tr>
[% FOREACH cart %]
<tr class="cartitem">
<td class="name">[% title %]</td>
<td><input class="quantity" name="quantity" size="3" value="[% quantity %]"></td>
<td class="price">[% price %]</td>
</tr>
[% END %]
</table>
```

1.2.5 Cart with HTML::Zoom

Cart: HTML::Zoom

```
use HTML::Zoom;

$cart = ...
$zoom = HTML::Zoom->from_file('cart.html');

$zoom = $zoom->select('.cartitem')->repeat_content([
  map { my $field = $_; sub {
     $_[0]->select('.name')->replace_content($field->{title});
     $_[0]->select('.quantity')->replace_content($field->{quantity});
     $_[0]->select('.price')->replace_content($field->{price});
     };
  } @$cart]);

$zoom = $zoom->select('.cost')->replace_content(49.95 + 34.99);

print $zoom->to_html();
```

1.2.6 Template Problems

Template Problems

- Mini language in HTML template
- Dynamic pages (border cases, errors, ...)

1.2.7 Cart with Template::Flute

Template::Flute Concept

- Specification
- Template
- Data or objects (iterator)

1.2.8 Specification

Template::Flute Specification (XML)

```
<specification name="cart" description="Cart">
<list name="cart" class="cartitem" iterator="cart">
<param name="name" field="title"/>
<param name="quantity"/>
<param name="price"/>
</list>
<value name="cost"/>
</specification>
```

Template::Flute Specification (Config::Scoped)

```
list cart {
   class = cartitem
   iterator = cart
}
param quantity {
   list = cart
}
param price {
   list = cart
}
param name {
   list = cart
   field = title
}
value cost {
   name = cost
}
```

1.2.9 Quellcode

Template::Flute Script (XML)

```
use Template::Flute;

my ($cart, $flute, %values);

$cart = ...
$values{cost} = ...

$flute = new Template::Flute(specification_file => 'cart.xml',
                                template_file => 'cart.html',
                                iterators => {cart => $cart},
                                values => \%values,
);

print $flute->process();
```

Template::Flute Script (Config::Scoped)

```
use Template::Flute;

my ($cart, $flute, %values);

$cart = ...
$values{cost} = ...

$flute = new Template::Flute(specification_file => 'cart.conf',
                                specification_parser => 'Scoped',
                                template_file => 'cart.html',
                                iterators => {cart => $cart},
                                values => \%values,
);

print $flute->process();
```

You are probably missing the $ sign in the HTML output, we see to that later.

1.3 Menu Example

1.3.1 Database table for menus

Menus: Database table

```
CREATE TABLE menus (
  code int NOT NULL auto_increment,
  name varchar(255) NOT NULL DEFAULT '',
  url  varchar(255) NOT NULL DEFAULT '',
  menu_name varchar(64) NOT NULL DEFAULT '',
  permission varchar(64) NOT NULL DEFAULT '',
  weight int NOT NULL DEFAULT 0,
  PRIMARY KEY(code),
  KEY(menu_name)
);
```

1.3.2 Specification

Menus: Specification

```
<specification name="menu" description="Menu">
<list name="menu" class="menu" table="menus">
<input name="name" required="1" field="menu_name"/>
<param name="label" field="name"/>
<param name="url"/>
</list>
</specification>
```

1.3.3 Template

The HTML template for the menu is really simple, because the styling can be done completely with CSS.

Menus: Template

```
<ul class="menu">
<li><a href="" class="url"><span class="label"></span></li>
</ul>
```

1.3.4 Script

Menus: Script

```
use Template::Flute;
use Template::Flute::Database::Rose;

$db_object = new Template::Flute::Database::Rose(dbname => 'temzoo',
    dbtype => 'Pg',
);

$flute = new Template::Flute(specification_file => 'menu.xml',
                      template_file => 'menu.html',
                      database => $db_object,
                     );

$flute->process({name => main});
```

2 Iterators

Template::Flute uses iterators to retrieve list elements and insert them into the document tree. This abstraction relieves us from worrying about where the data actually comes from. We basically just need an array of hash references and an iterator class with a next and a count method. For your convenience you can create an iterator from Template::Flute::Iterator very easily.

Iterators

- next method
- count method
- hash reference as return value

2.1 Template::Flute::Iterator

Template::Flute::Iterator

```
use Template::Flute::Iterator;

Template::Flute::Iterator->new($cart);
```

2.2 Subclassing Template::Flute::Iterator

Subclassing Template::Flute::Iterator

```
package MyApp::Iterator;

use base 'Template::Flute::Iterator';

sub new {
    ...
    $self->seed([...]);
    return $self;
}
```

3 Elements

Elements

- value
- list
- param
- input
- container
- form
- i18n
- sort

3.1 List with alternating rows

Lists with alternating rows

```
<table class="cart">
<tr class="cartheader">
<th>Name</th><th>Quantity</th><th>Price</th>
</tr>
<tr class="cartitem odd">
<td class="name">Perl 6</td>
<td><input class="quantity" name="quantity" size="3" value="10"></td>
<td class="price">$$$</td>
</tr>
<tr class="cartitem even">
<td class="name">Pro Git</td>
<td><input class="quantity" name="quantity" size="3" value="10"></td>
<td class="price">$$$</td>
</tr>
</table>
```

3.2 Filter and Sort

There are two types of filters for lists: global filters and parameter filters. Global filters are applied to the complete record of a list element and can be used to skip list items. Parameter filters are applied to a single value in a list element record.

3.3 Parameter Filter

Filter: Specification

```
<specification name="menu" description="Menu">
<list name="menu" class="menu" table="menus">
<input name="name" required="1" field="menu_name"/>
<param name="label" field="name"/>
<param name="url" target="href" filter="link"/>
</list>
</specification>
```

3.4 Filter function

Filter: Function

```
sub link_filter {
    my $page = shift;
    my $url;

    $url = ...

    return $url;
}

$flute = new Template::Flute(specification_file => 'menu.xml',
                             template_file => 'menu.html',
                             database => $db_object,
                             filters => {link => \&link_filter},
                            );
```

3.4.1 Global Filter

Global Filter

```
<specification name="menu" description="Menu">
<filter name="acl" field="permission"/>
<list name="menu" class="menu" table="menus">
<input name="name" required="1" field="menu_name"/>
<param name="label" field="name"/>
<param name="url" target="href" filter="link"/>
</list>
</specification>
```

3.4.2 Specification with sort

Sort: Specification

```
<specification name="menu" description="Menu">
<list name="menu" class="menu" table="menus">
<sort name="default">
<field name="weight" direction="desc"/>
<field name="code" direction="asc"/>
</sort>
<input name="name" required="1" field="menu_name"/>
<param name="label" field="name"/>
<param name="url" target="href" filter="link"/>
</list>
</specification>
```

3.5 I18N

I18N support is very basic right now. You write a function for translating text inside the HTML template and instantiate an Template::Flute::I18N with a reference to this function.

I18N

```
sub translate {
    my $text = shift;

    ...

    return $text;
}

$i18n = new Template::Flute::I18N (\&translate);

$flute = new Template::Flute(specification_file => 'menu.xml',
                             template_file => 'menu.html',
                             database => $db_object,
                             i18n => $i18n,
                            );
```

3.5.1 I18N: Lookup Keys

You can override the text in the HTML template passed to the translation function with a lookup key in the specification.

I18N: Lookup Keys

```
<i18n name="returnurl" key="RETURN_URL"/>
```

3.6 Forms

Forms: Specification

```
<specification name='search' description=''>
<form name='search'>
<field name='searchterm'/>
<field name='searchsubmit'/>
</form>
</specification>
```

3.6.1 Manipulating Forms

The Template::Flute::Form class provides a number of methods to manipulate the output of forms in the resulting HTML:

Forms: Manipulating

set_action Changing form action

set_method Changing form method (GET, POST)

fill Fill form fields

4 PDF

PDF generation starts just the same way as HTML template processing. In fact, it might make sense to use the same template for display in the browser and for producing the PDF document.

The conversion is running through 3 passes. First the position and sizes of the boxes are calculated. Second the boxes are partitioned throughout the pages in the PDF document.

4.1 HTML to PDF

HTML to PDF

- HTML template processing
- PDF conversion (PDF::API2)
    1. calculate
    2. partition
    3. render
- Inline CSS

PDF: Code

```
$flute = new Template::Flute (specification_file => 'pdf.xml',
                              template_file => 'pdf.html',
                              values => \%values);
$flute->process();

$pdf = new Template::Flute::PDF (template => $flute->template(),
                                 file => 'invoice.pdf');
$pdf->process();
```

4.2 Import

PDF: Import

```
$import{file} = 'shippinglabel.pdf';
$import{scale} = 0.8;
$import{margin} = {left => '3mm', top => '6mm'};

$pdf = new Template::Flute::PDF (template => $flute->template(),
                                 file => 'invoice.pdf',
                                 import => \%import);
```

5 Dancer

Dancer Example

```
use Dancer;

get '/' => sub {
    return 'Hello world';
};

dance;
```

5.1 Fruits Demo

Fruits Demo

```
dancer -a Fruits
$EDITOR Fruits/lib/Fruits.pm
```

(see next slide)

```
Fruits/bin/app.pl
```

Fruits Demo

```
package Fruits;

prefix '/fruits';

# route for image files
```

```
get '/*.jpg' => sub {
    my ($name) = splat;

    send_file "images/$name.jpg";
};

# route for fruits page
get qr{/?(?<page>.*)}  => sub {
    template 'fruits';
};
```

5.2 Dancer & Template::Flute

Dancer & Template::Flute

```
template: "template_flute"

engines:
  template_flute:
    iterators:
      fruits:
        class: JSON
        file: fruits.json
```

6 Conclusion

6.1 Use Cases

Current and Future Use Cases

- Very Large Product Lists
- Shop Backend
  - Product Editor
  - Product Search & Replace
- PDF Invoices
- Template Engine for Interchange

6.2 Roadmap

Roadmap

- Documentation
- Tests
- Conditions

- Empty lists, number of results
- Selected items
- Paging
- Trees

6.3 The End

The End

Git git://git.linuxia.de/temzoo.git

CPAN `http://search.cpan.org/dist/Template-Flute/`

`http://search.cpan.org/dist/Template-Flute-PDF/`

`http://search.cpan.org/dist/Dancer-Template-TemplateFlute/`

Talk `http://www.linuxia.de/talks/hhmongers2011/tf-hhmongers2011-beamer.pdf`

Questions ???