Template::Flute - Modern designer-friendly HTML templating engine

Stefan Hornburg (Racke)
racke@linuxia.de

Perl Workshop in Israel, 28th February 2012, Ramat Gan

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

Template Example

Name Quantity Price

Total

Template Example: Data

```
@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},
);
```

Template Example: Output

Name	Quantity	Price
The Pragmatic Programmer	1	$ 49.95
Pro Git	1	$ 34.99
	Total	$ 100

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>
```

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="[% quanti
<td class="price">[% price %]</td>
</tr>
[% END %]
<<tr class="cartheader"><th colspan="2"></th><th>Total</th></tr>
<tr>
<td colspan="2"></td><td class="cost">[% total %]</td>
</tr>
</table>
```

Common Problems

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

Concept

- HTML Template
- Specification
- Data or objects (iterator)

Basic Elements

- Variables
- Conditions
- Loops

Variables

HTML

```
<span class="email">foo@bar.com</span>
```

Specification

```
<value name="email" class="email"/>
```

Code

```
$flute->process(values => {email => 'racke@linuxia.de'});
```

Output

```
<span class="email">racke@linuxia.de</span>
```

Variables

HTML

```
<span id="email">foo@bar.com</span>
```

Specification

```
<value name="email" id="email"/>
```

Code

```
$flute->process(values => {email => 'racke@linuxia.de'});
```

Output

```
<span class="email">racke@linuxia.de</span>
```

Conditions

HTML

```
<div class="warnings">What's up?</div>
```

Specification

```
<container name="warnings" class="warnings" value="message">
<value name="message" class="warnings"/>
</container>
```

Code

```
$flute->process(values => {message => 'No coffee available'});
```

Output

```
<div class="warnings">No coffee available</div>
```

List Example

HTML

```
<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>
```

Spezifikation

```
<list name="cart" class="cartitem" iterator="cart">
<param name="name" field="title"/>
<param name="quantity"/>
<param name="price"/>
</list>
```

List Example

Code

```
$flute = Template::Flute->new(template => $template,
                              specification => $specification,
                              iterators => {cart => \@cart});

$output = $flute->process;
```

Iterators

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

Template::Flute::Iterator

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

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

Subclassing Template::Flute::Iterator

```
package MyApp::Iterator;

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

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

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>
```

Hints

- Targets
- Expressions
- Operators

Explicit and implicit targets

Link (Template::Toolkit)

```
<a href="[% login_url %]" class="url">Login</li>
```

Link (Template::Flute Specification)

```
<value name="url" target="href"/>
```

Hidden form field (Template::Toolkit)

```
<input type="hidden" name="sku" class="sku" value="[% sku %]">
```

Hidden form field (Template::Flute Specification)

```
<value name="sku" class="sku"/>
```

Expressions

```
<container name="summary" value="wishlist_name&&!wishlist_page">
<value name="wishlist-name" field="wishlist_name"/>
<value name="total" class="wishlist-sum-total" filter="currency"/>
<list name="cart_collections" class="wishlist-collections-line" itera
<param name="sku" class="btn-remove" target="value"/>
<param name="name" class="wishlist-title"/>
<param name="quantity" class="wishlist-qty"/>
<param name="wishlist-title" field="sku" target="href" op="append"/>
</list>
</container>
```

Operators

- append
- toggle
- hook

Filters

Values

```
<value name="total_cost" filter="currency"/>
```

Lists

```
<param name="price" filter="currency"/>
```

Filter types

- Built-in filters
- Custom filter classes
- Custom filter functions

Builtin filters

- date
- currency
- upper
- eol

Custom filter class

```
package MyApp::Filter;

sub filter {
    my ($self, $value) = @_;

    return $value;
}

1;
```

Custom 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',
   filters => {link => \&link_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>
```

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,
  );
```

I18N: Lookup Keys

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

Forms: Specification

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

Forms: Manipulating

set_action	Changing form action
set_method	Changing form method (GET, POST)
fill	Fill form fields

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();
```

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);
```

Dancer Example

```
use Dancer;

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

dance;
```

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';
};
```

Dancer & Template::Flute

```
template: "template_flute"

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

Current and Future Use Cases

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

Roadmap

- Documentation & Tests
- Empty lists, number of results
- Paging
- Trees
- CSS selectors

The End

Git	git://github.com/racke/Template-Flute.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/ilpw2012/tf-beamer.pdf
Questions	???