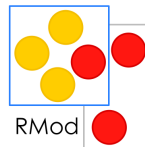


Pillar, one format for all supports

Thibault Arloing Yann DUBOIS

26 août 2016



- 1 Pillar
- 2 Pillar use cases
- 3 What's new in Pillar 4.0
- 4 Make utilisation easier
- 5 Demo Time
- 6 Conclusion

- LaTeX
 - Complicated
 - Hard to debug
 - Hard to convert to HTML
- Markdown
 - Incomplete
 - Incompatible Implementation
 - Few output formats

Why Pillar ?

We wanted something :

- Easier than LaTeX
- More complete than Markdown

Pillar

- One input, many outputs (e.g. HTML, LaTeX, ePub)

Pillar - EPub export

EnterprisePharo

Simple Web applications

Getting Started

Differences between Teapot and other Web Frameworks

A REST Example, Showing some CRUD Operations

Route

Parameters in URLs

Using Regular Expressions

How are Routes Matched?

Aborting

Transforming Output from Actions

Response Transformers

Before and After Filters

Error Handlers

Serving Static Files

Conclusion

Saying Hello World

Debugging our Web App

Serving an HTML Page With an Image

Serving an Image



Teapot is not a singleton and doesn't hold any global state. You can run multiple Teapot servers inside the same image with their state being isolated from each other.

- There are no thread locals or dynamically scoped variables in Teapot. Everything is explicit.
- It doesn't rely on annotations or pragmas, the routes are defined programmatically.
- It doesn't instantiate objects (e.g. "web controllers") for you. You can hook http events to existing objects, and manage their dependencies as required.

A REST Example, Showing some CRUD Operations

Before getting into the details of Teapot. Here is a simple example for managing books. With the following code, we can list books, add a book and delete a book.

```
| books teapot |
books := Dictionary new.
teapot := Teapot configure: {
  #defaultOutput -> #json. #port -> 8080. #debugMode -> true }.
teapot
  GET: '/books' -> books;
  PUT: '/books/<id>' -> [ :req | | book |
    book := {'author' -> (req at: #author).
    'title' -> (req at: #title)} asDictionary.
    books at: (req at: #id) put: book ];
  DELETE: '/books/<id>' -> [ :req | books removeKey: (req at:
#id) ];
  exception:
    keyNotFound -> (TeaResponse notFound body: 'No such book!');
```

```
start.
```

Now you can create a book with ZNClient or your web client as follows:

```
ZnClient new
  url: 'http://localhost:8080/books/1';
  formAt: 'author' put: 'SquareBracketAssociates';
  formAt: 'title' put: 'Pharo For The Enterprise'; put
```

You can also list the contents using <http://localhost:8080/books> For a more complete example, study the Teapot-Library-Example package.

Now that you get the general feel of Teapot, let us see the key concepts.

Route

The most important concept of Teapot is the Route. The template for route definitions is as follows:

```
Method : '/url'/'pattern'/'<param' -> Action
```

A route has three parts:

- an HTTP method (GET, POST, PUT, DELETE, HEAD, TRACE, CONNECT, OPTIONS, PATCH),
- an URL pattern (i.e. /n1, /users/<name>, /foo/*/bar/*, or a regular expression),
- an action (a block, message send or any object).

Here is another example:

```
Teapot on
```

- One input, many outputs (e.g. HTML, LaTeX, ePub)
- Slides / Books / Websites
- Textual syntax
- Easy to Extend
- Easy to use

Pillar - Easy syntax

!Example

This is an example of Pillar file.

```
*Link>http://pharo.org*
```

```
-Unordered Item
```

```
-Unordered Item
```

```
#Ordered Item
```

```
#Ordered Item
```

```
[[[language=smalltalk|label=How to say Hello
```

```
Transcript show: 'Hello !'.
```

```
]]]
```

```
!Harder !Better
```

```
|Faster |Stronger
```

```
|More Than |Ever
```

```
+Figure>file://path/to/the/file.png|width=80+
```

Advanced features

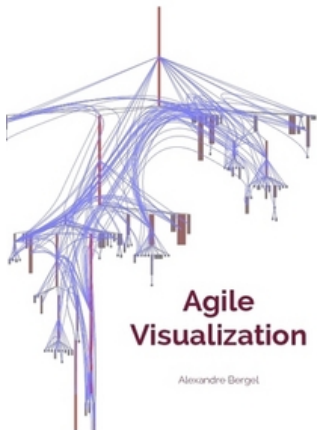
- `${}` annotation syntax
- Slides `slide :Slide name`
- `{myAnnotation :arg1=1|arg2=2}`
- Consistency in syntax `[[[language=smalltalk|arg1=1|arg2=2]]]`
- Evaluate a smalltalk script

Pillar use cases

Books written with Pillar

- Enterprise Pharo : A Web Perspective
- Agile Virtualization
- Pharo By Example

Books written with Pillar



Presentations

- Beamer
- DeckJs

This presentation itself is written in Pillar

Other use cases

- Ecstatic : Static Websites, Guillermo Polito / Stéphane Ducasse (<http://guillep.github.io/ecstatic>)
- PillarHub, Mike Filonov (<http://pillarhub.pharocloud.com/>)

What's new in Pillar 4.0

- Dissociate from the shape
- Easy to use
- Provide from formatting mistakes

Structures - How does it work

```
[[[structure=exercice  
{  
  "question":"The question is here",  
  "answer":"42"  
}  
]]]
```



PRScriptStructureTransformer



PRStructure
(
 'question' -> 'The question here',
 'answer' -> '42')



PRDefinitionListRenderer



```
PRDefinitionList  
- PRTermItem('question')  
- PRDataItem('The question is here')  
- PRTermItem('answer')  
- PRDataItem('42')
```

Pillar 4.0 - Other features

- Footnotes $\${footnote :Text to put in the footnote}\$$
- Citations $\${cite :REF1713}\$$
- Header Capitalization
- Hideable Scripts

Make utilisation easier

Hell on boot project

- Where is my configuration file ?
- Where are templates ?
- I am bored of copy/paste parts of configuration file
- Where did I put my pillar files ?

Pillar Archetypes - Skeleton maker

- Simple utilisation
- Build a project base with example files
- Many archetypes (Book / Slides / Beginner)

```
./pillar archetype book
```

Pillar Archetype - How does it work ?



Makefile Archetypes

- Makefile adapted to each Archetypes
- Separated Makefiles

- Begin with a pillar file

first.pillar

```
{  
  "metadata": {  
    "title" : "Say Hello World"  
  }  
}  
!Hello World
```

- Pass through Pillar

```
$(OUTPUTDIRECTORY)/%.html.json: %.pillar copySupport  
./pillar export --to="html" --outputFile=$@ $<
```

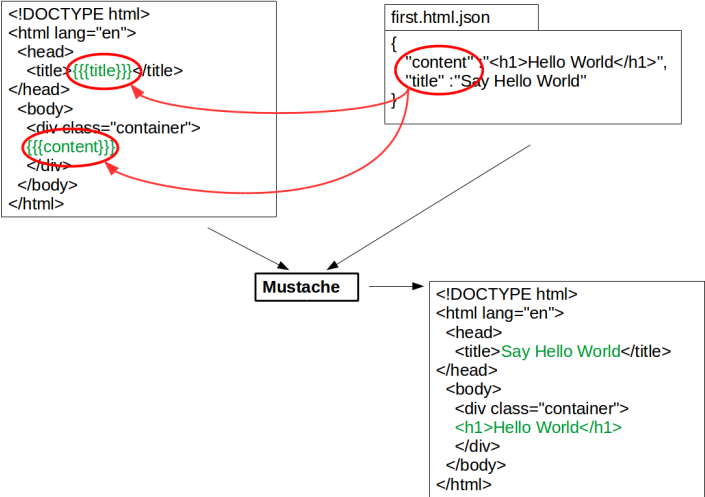
first.html.json

```
{  
  "content" : "<h1>Hello World</h1>",  
  "title" : "Say Hello World"  
}
```

- Pass through a template System

```
./mustache --data=$< --template=${HTMLTEMPLATE} > $@
```

Makefile Job



Demo Time

Conclusion

Future Works

- Pharo comments in Pillar
- Mobi Exporter
- Math expressions in LaTeX
- We have a lot of suggestions from community

Conclusion

- Easy to Extend
- Easy Syntax
- Lot of output
- Easier Compilation
- Possibility to use other tools for templating

For more informations

<https://ci.inria.fr/pharo-contribution/job/EnterprisePharoBook/lastSuccessfulBuild/artifact/book-result/PillarChap/Pillar.html>