

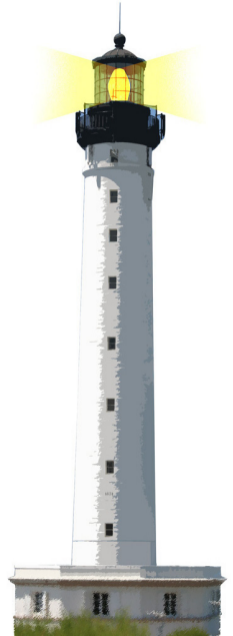
Seaside: An Innovative Web Application Framework

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W4S08



<http://www.pharo.org>



Seaside

- A powerful, innovative and flexible framework
- Dedicated to build complex Web applications
- Live coding and debugging
- Support reusable Web components
- Secure by default
- Web 2.0 support (Ajax, Reef, ...)
- REST integration

seaside 

Books and Tutorials

- Seaside `http://www.seaside.st`
- Seaside book `http://book.seaside.st`
- Seaside tutorial
`http://www.swa.hpi.uni-potsdam.de/seaside/`
- Seaside tutorial
`http://seaside.gemtalksystems.com/tutorial.html`
- TinyBlog tutorial
- Community: register to seaside mailing list and ask questions



Seaside Little History

- Developed by A. Bryant and J. Fitzell
- Enhanced by L. Renggli and P. Marshall
- In production since 2002
- Actively maintained by J. Brichau, S. Eggermont (web site under full rewrite)
- Foundation of many Pharo success stories
- <http://www.pharo.org/success>

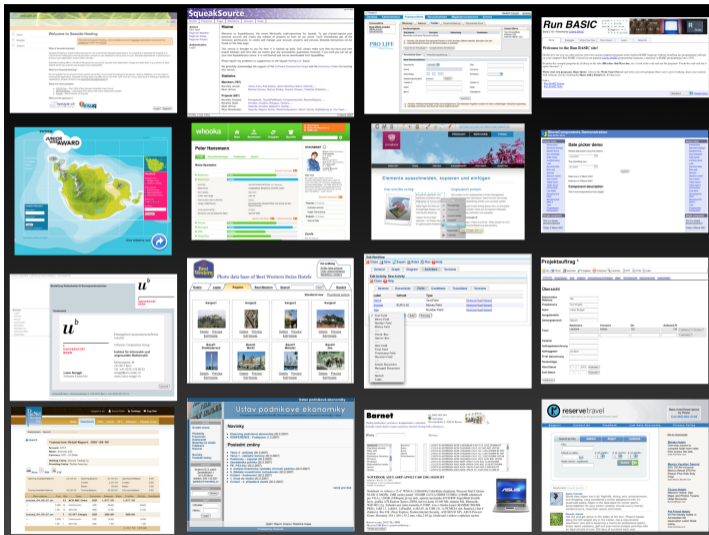


Seaside in a Nutshell

- Define reusable and stateful components
- Use a DSL for rendering components
- Compose components
 - build coarser-grained components by encapsulation
 - schedule components with `call:` and `answer:` messages
- A web application is just a root component
- Debug your application on the fly
- Use metadata to generate forms



Seaside in Production Since 2002



Quuve - debrispublishing.com

The screenshot displays the Quuve software interface, which is used for portfolio management and analysis. The interface is divided into several main sections:

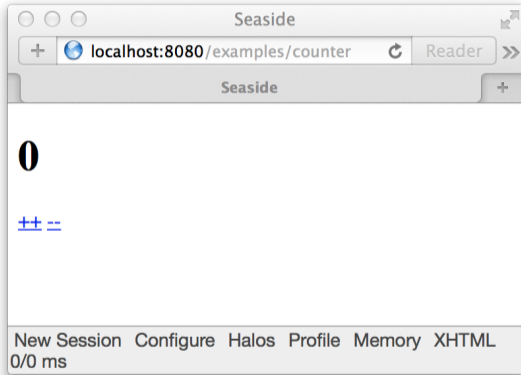
- Top Panel:** Includes navigation tabs (Home, Charts, Portfolio, Research, Reports) and a search bar.
- Left Panel:** Contains a 'Data Values' table with columns for 'Date' and 'Value'. It lists various financial metrics such as 'Revenue', 'Expenses', and 'Net Income' over time.
- Center Panel:** Features a large data table with columns for 'Symbol', 'Company Name', 'Sector', 'Market Cap', 'P/E Ratio', and 'Yield'. It lists numerous stocks and their associated financial data.
- Right Panel:** Contains a 'Welcome to Quuve' message and a 'Get Started' button. It also includes a 'Copyright © 2007-2016 Debris Publishing, Inc. All rights reserved.' notice.
- Bottom Panel:** Includes a 'PER BASIC RATIOS' table with columns for 'Year' and 'Ratio'. It lists various financial ratios such as 'P/E Ratio', 'P/B Ratio', and 'Dividend Yield' over time.
- Bottom Right Panel:** Contains a 'Charts' section with several pie charts and a 'Portfolio Allocations' table. The pie charts show the distribution of assets across different sectors and regions. The table lists various asset classes and their corresponding allocations.

Seaside Components

- A component is:
 - an instance of a subclass of `WComponent`
 - a reusable and stateful part of a **Web page**
 - rendered in HTML (`<div>`)
- A Web application has a root component

`WAdmin` register: `WCounter` asApplicationAt: 'counter'.

The Counter Web Application



WACounter

```
WACounter subclass: #WACounter  
instanceVariableNames: 'count'  
classVariableNames: ''  
package: 'Seaside-Examples-Misc'.
```

```
WACounter >> initialize  
super initialize.  
count := 0
```

```
WACounter >> increase  
count := count + 1
```

```
WACounter >> decrease  
count := count - 1
```

From Components to Valid HTML

- All components respond to `renderContentOn`:
- This method converts a component to valid HTML
- This message is automatically sent to components by Seaside



HTML Rendering

- `renderContentOn`: is dedicated to HTML generation
- parameter named `html` (`WAHtmlCanvas`) defines a DSL like API to generate valid HTML

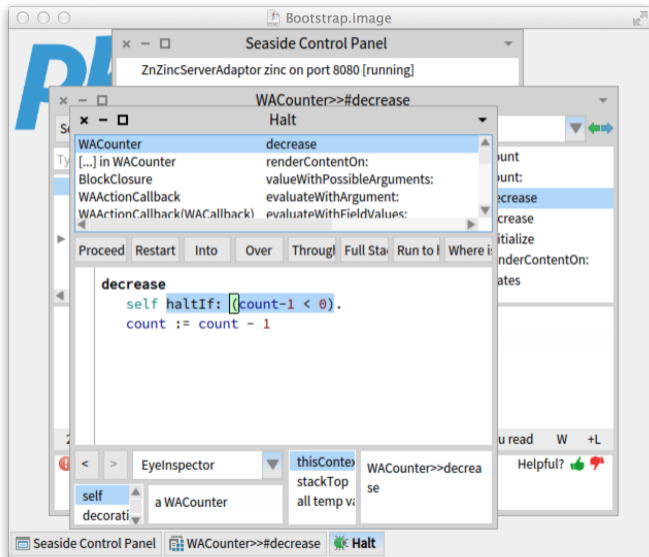
```
WACounter >> renderContentOn: html
html heading: count.
html anchor
  callback: [ self increase ];
  with: '++'.
html space.
html anchor
  callback: [ self decrease ];
  with: '--'
```



Live Debugging

```
WACounter>>decrease  
self haltIf: (count-1 < 0).  
count := count - 1
```

Walking the Application Stack



Back Button

Pressing the **back button** of the browser desynchronizes server and client

Example:

- Increment the counter 5 times (count = 5)
- Press the **back button** => the displayed value is 4
- Increment the counter => the displayed value is 6

How to make it work properly?



A Counter Dealing with Back Button

Just declare the component state to be preserved

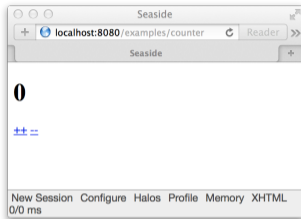
```
WCounter >> states  
  ^ Array with: self
```

Plain Code in Callbacks

```
WCounter >> renderContentOn: html  
html heading: count.  
html anchor  
  callback: [  
    count odd  
    ifTrue: [ self increase ]  
    ifFalse: [  
      self inform: 'Even number!'.  
      count := count + 2]  
  ];  
with: '++'.  
html space.  
html anchor  
  callback: [ self decrease ];  
with: '--'
```

Callback Execution

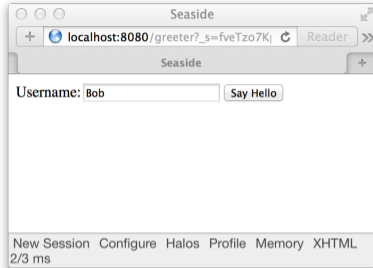
Pressing ++



shows



A Greeter Application



Callbacks with the User Value

A Greeter component

```
Greeter >> renderContentOn: html
  html form: [
    html text: 'Username:'.
    html textInput
      callback: [ :value | username := value ].
    html submitButton
      callback: [ self inform: 'Hi ', username ];
      text: 'Say Hello' ].
```

Did you see?!

- **No** manual request parsing
- **No** XML configuration files
- **No** file/page
 - don't think in terms of pages
 - use components
- **No** hardcoding of next page
- **Live Debugging**
 - use the debugger to modify objects and proceed to generate the HTML response



Conclusion

- A Web application = a root component
- A component renders itself in HTML (renderContentOn:)
- An extensible DSL helps to easily generate HTML



A course by



and



in collaboration with



Inria 2016

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France

<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>