5. Seaside

Roadmap

> Introduction
  - Web applications / Overview
  - Installation
> Control Flow
> Components
> Composition

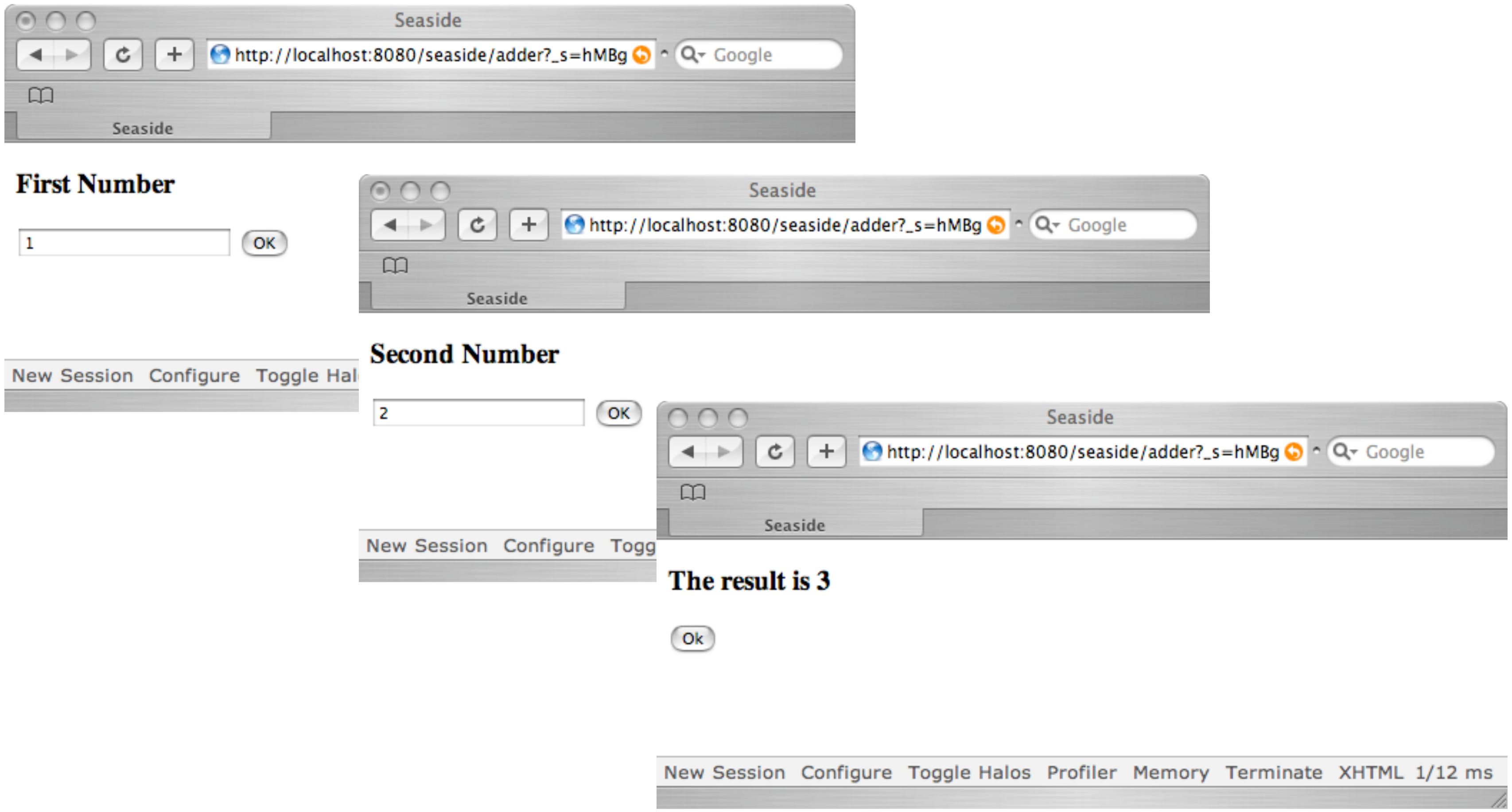
Original lecture notes by Lukas Renggli

Roadmap

> Introduction
  — **Web applications / Overview**
  — Installation
> Control Flow
> Components
> Composition

Introduction: Web Applications

Example: Adding two numbers



Seaside

http://localhost:8080/seaside/adder?_s=hMBg

First Number

1

OK

New Session Configure Toggle Hal

Seaside

http://localhost:8080/seaside/adder?_s=hMBg

Second Number

2

OK

New Session Configure Togg

Seaside

http://localhost:8080/seaside/adder?_s=hMBg

The result is 3

Ok

New Session Configure Toggle Halos Profiler Memory Terminate XHTML 1/12 ms

What is going on?

```
<form action="second.html">
   <input type="text" name="value1">
   <input type="submit" name="OK">
</form>
```

First.hml

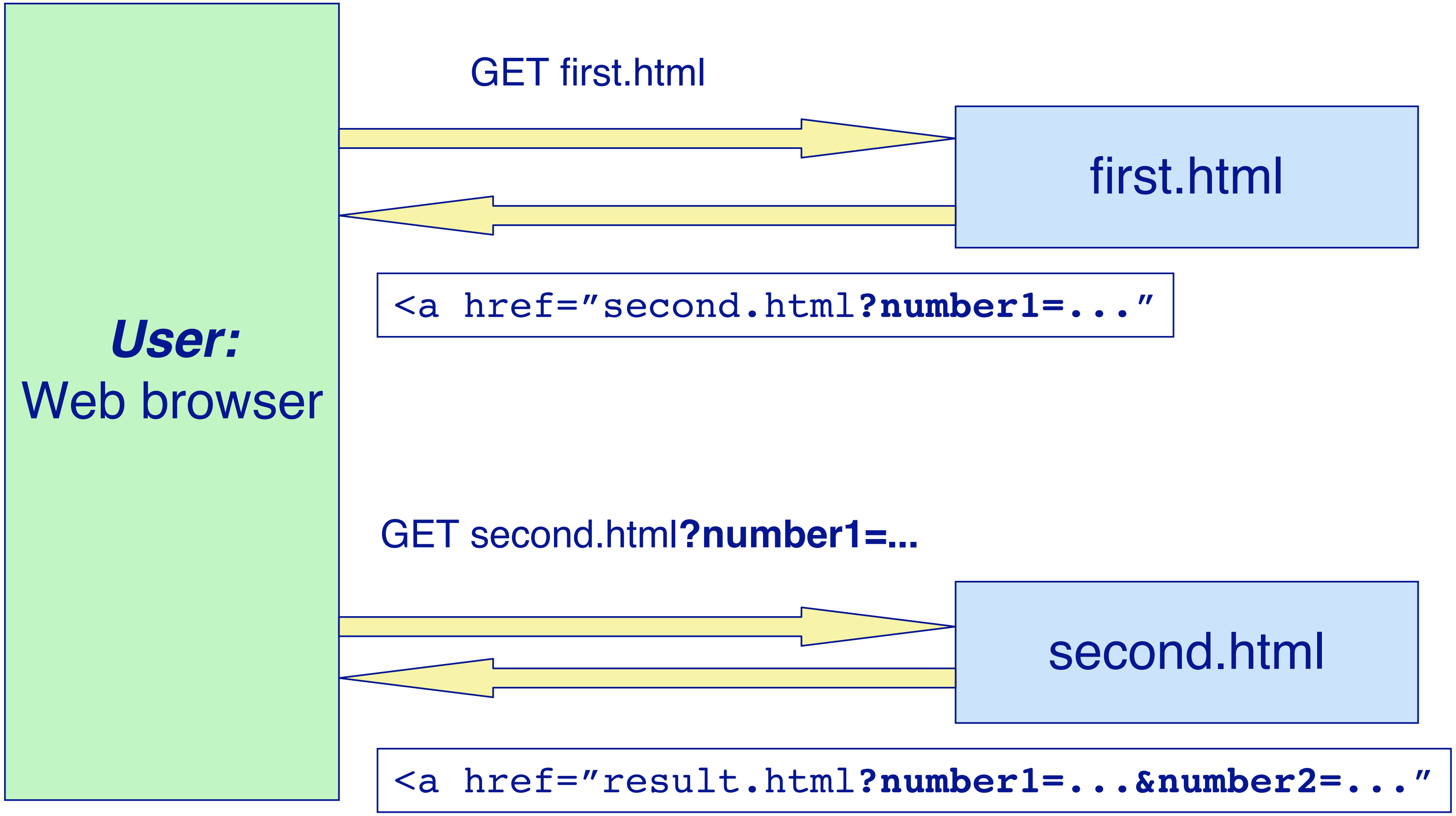
```
<form action="result.html">
   <input type="hidden"
               name="value1" value="<% value1 %>">
</form>
```

second.hml

```
<p>
   <% value1 + value2 %>
</p>
```

result.hml

Control Flow: HTTP request-response



```
<a href="second.html?number1=..."
```

```
<a href="result.html?number1=...&number2=..."
```

Something is wrong…

- Control-flow quite arcane
  - Remember GOTO?
  - We do not care about HTTP!
- How to debug that?
- And what about
  - Back button?
  - Copy of URL (second browser)?

What we want

> Why not this?

```
go
   |number1 number2 |

   number1 := self request: ‘First Number’.
   number2 := self request: ‘Second Number’.

   self inform: ‘The result is ‘,
               (number1 + number2) asString
```

Seaside: Features

- Sessions as continuous piece of code
- XHTML/CSS building
- Callback based event model
- Composition and reuse
- Debugging and Development tools

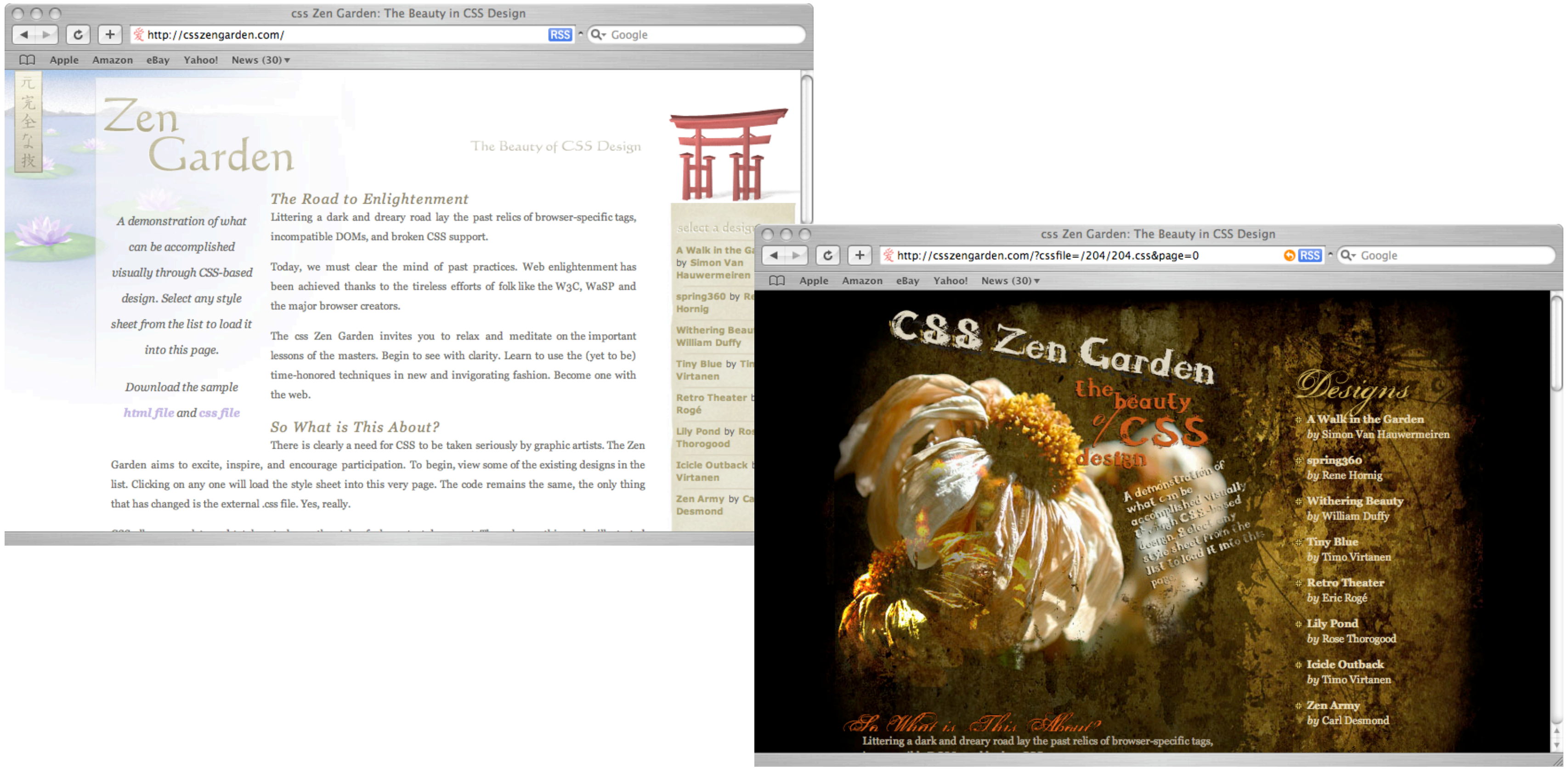
XHTML Building

```
html div id: ‘title’; with: ‘Title’
html div id: ‘list’; with: [
   html span class: ‘item’; with: ‘Item 1’.
   html span class: ‘item’; with: ‘Item 2’.
]
```

```
<div id=“title”>Title</div>
<div if=“list”>
   <span class=“item”>Item 1</span>
   <span class=“item”>Item 2</span>
</div>
```

CSS

> CSS Zengarden: http://csszengarden.com


css Zen Garden: The Beauty in CSS Design
http://csszengarden.com/
Apple Amazon eBay Yahoo! News (30)
Zen Garden
The Beauty of CSS Design
select a design
css Zen Garden: The Beauty in CSS Design
http://csszengarden.com/?cssfile=/204/204.css&page=0
CSS Zen Garden
the beauty of CSS design
Designs

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

Download the sample html file and css file

- A Walk in the Garden by Simon Van Hauwermeiren
- spring360 by Rene Hornig
- Withering Beauty by William Duffy
- Tiny Blue by Timo Virtanen
- Retro Theater by Eric Rogé
- Lily Pond by Rose Thorogood
- Icicle Outback by Timo Virtanen
- Zen Army by Carl Desmond

So What is This About?

Littering a dark and dreary road lay the past relics of browser-specific tags,

Callback Event Model

```
Example3>>renderContentOn: html
              html form: [
              html submitButton
              callback: [ self inform: ‘Hello’ ];
              text: ‘Say Hello’ ]
```

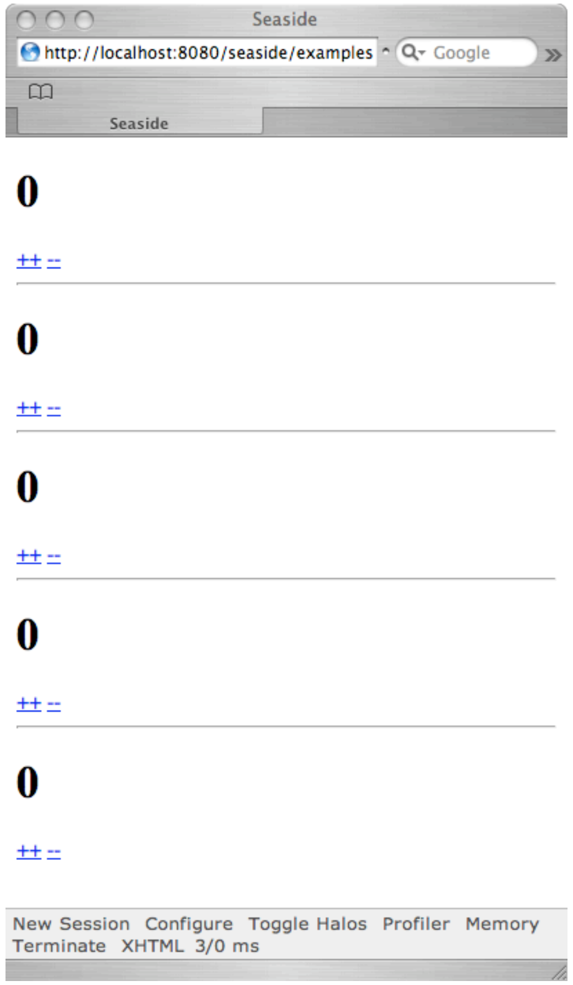
```
....
<form action="/seaside/example2" method="post">
<input type="hidden" name="_s" value="JBbTXBnPaTLOjcjI" class="hidden"/>
<input type="hidden" name="_k" value="FFQrpnBg" class="hidden" />
<input type="submit" name="1" value="Say Hello" class="submit" />
</form>
....
```

Composition + Reuse

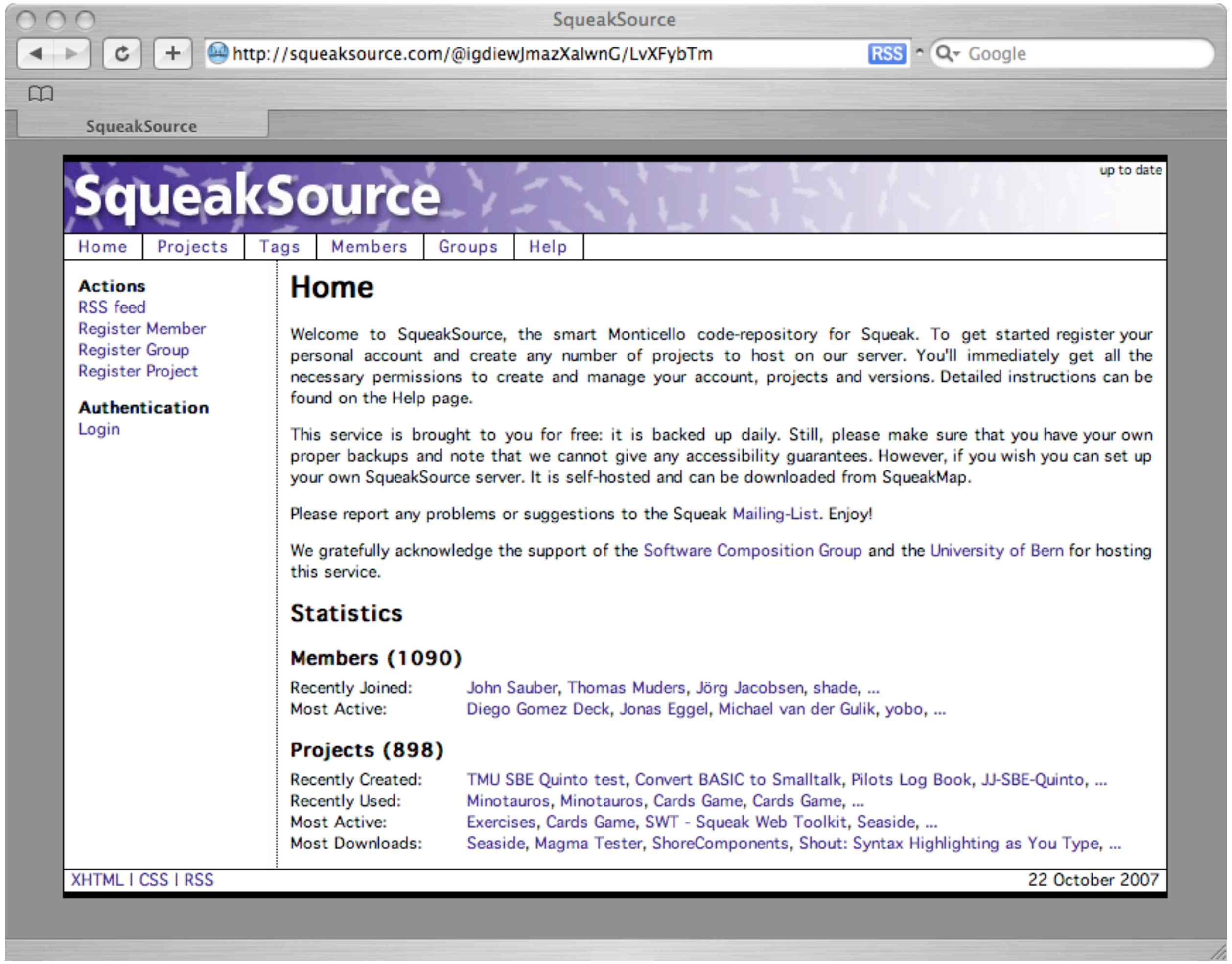
Example:

Multicounter

More later!


Seaside
http://localhost:8080/seaside/examples
Google
Seaside
0
++ --
0
++ --
0
++ --
0
++ --
0
++ --
New Session Configure Toggle Halos Profiler Memory
Terminate XHTML 3/0 ms

Example: SqueakSource


SqueakSource
http://squeaksource.com/@igdiewJmazXalwnG/LvXFybTm
RSS
Google
SqueakSource
SqueakSource
up to date
Home | Projects | Tags | Members | Groups | Help

Actions
RSS feed
Register Member
Register Group
Register Project

Authentication
Login

Home

Welcome to SqueakSource, the smart Monticello code-repository for Squeak. To get started register your personal account and create any number of projects to host on our server. You'll immediately get all the necessary permissions to create and manage your account, projects and versions. Detailed instructions can be found on the Help page.

This service is brought to you for free: it is backed up daily. Still, please make sure that you have your own proper backups and note that we cannot give any accessibility guarantees. However, if you wish you can set up your own SqueakSource server. It is self-hosted and can be downloaded from SqueakMap.

Please report any problems or suggestions to the Squeak Mailing-List. Enjoy!

We gratefully acknowledge the support of the Software Composition Group and the University of Bern for hosting this service.

Statistics

Members (1090)

Recently Joined:	John Sauber, Thomas Muders, Jörg Jacobsen, shade, ...
Most Active:	Diego Gomez Deck, Jonas Eggel, Michael van der Gulik, yobo, ...

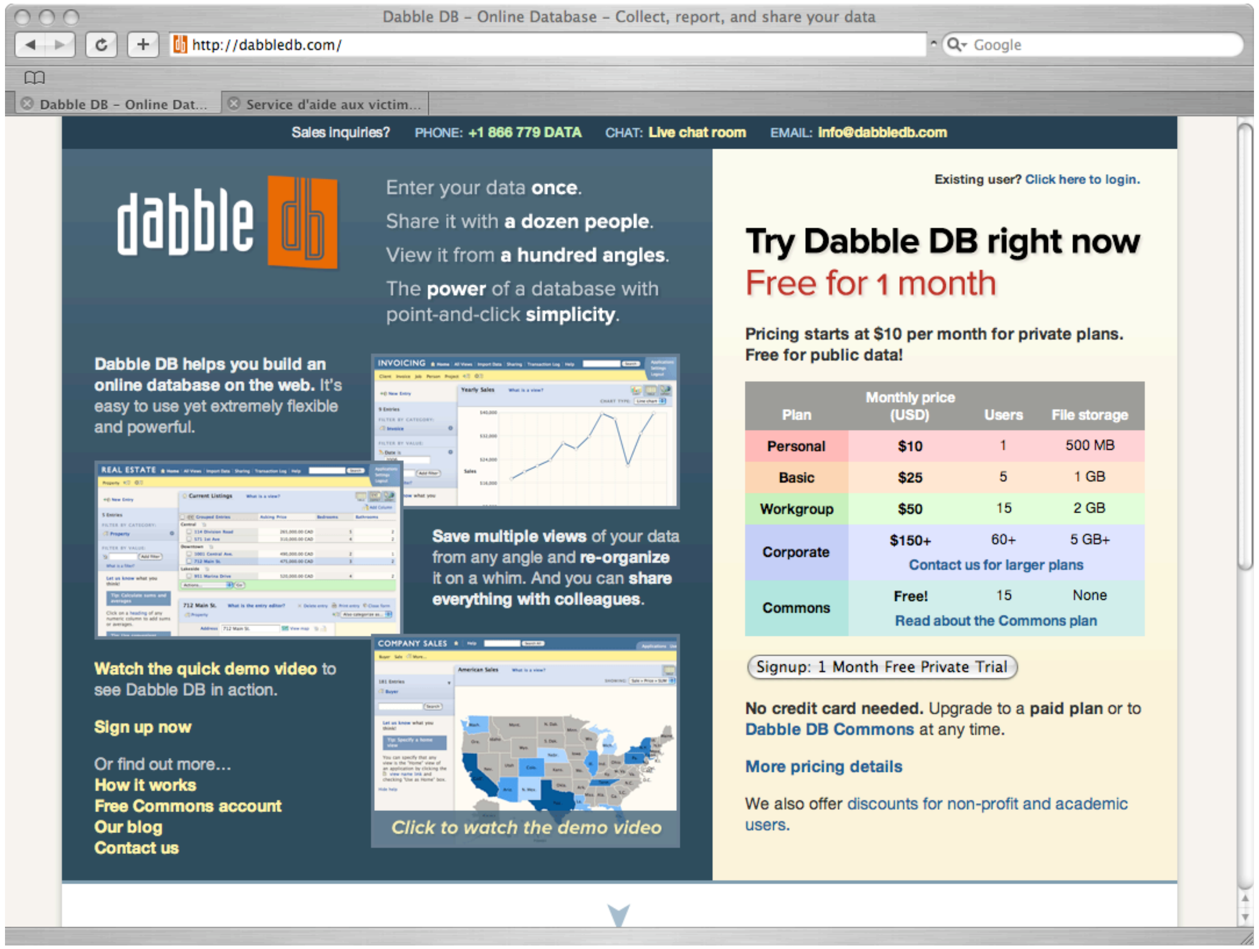
Projects (898)

Recently Created:	TMU SBE Quinto test, Convert BASIC to Smalltalk, Pilots Log Book, JJ-SBE-Quinto, ...
Recently Used:	Minotauros, Minotauros, Cards Game, Cards Game, ...
Most Active:	Exercises, Cards Game, SWT - Squeak Web Toolkit, Seaside, ...
Most Downloads:	Seaside, Magma Tester, ShoreComponents, Shout: Syntax Highlighting as You Type, ...

XHTML | CSS | RSS — 22 October 2007

Example: DabbleDB

Dabbledb.com



Dabble DB – Online Database – Collect, report, and share your data

http://dabbledb.com/

Dabble DB – Online Dat... | Service d'aide aux victim...

Sales inquiries? PHONE: +1 866 779 DATA CHAT: Live chat room EMAIL: info@dabbledb.com

Enter your data **once**.
Share it with **a dozen people**.
View it from **a hundred angles**.
The **power** of a database with point-and-click **simplicity**.

Dabble DB helps you build an online database on the web. It's easy to use yet extremely flexible and powerful.

Save multiple views of your data from any angle and **re-organize** it on a whim. And you can **share everything with colleagues**.

Watch the quick demo video to see Dabble DB in action.

Sign up now

Or find out more...
How it works
Free Commons account
Our blog
Contact us

Click to watch the demo video

Existing user? Click here to login.

Try Dabble DB right now
Free for 1 month

Pricing starts at $10 per month for private plans.
Free for public data!

Plan	Monthly price (USD)	Users	File storage
Personal	$10	1	500 MB
Basic	$25	5	1 GB
Workgroup	$50	15	2 GB
Corporate	$150+	60+	5 GB+
	Contact us for larger plans		
Commons	Free!	15	None
	Read about the Commons plan		

Signup: 1 Month Free Private Trial

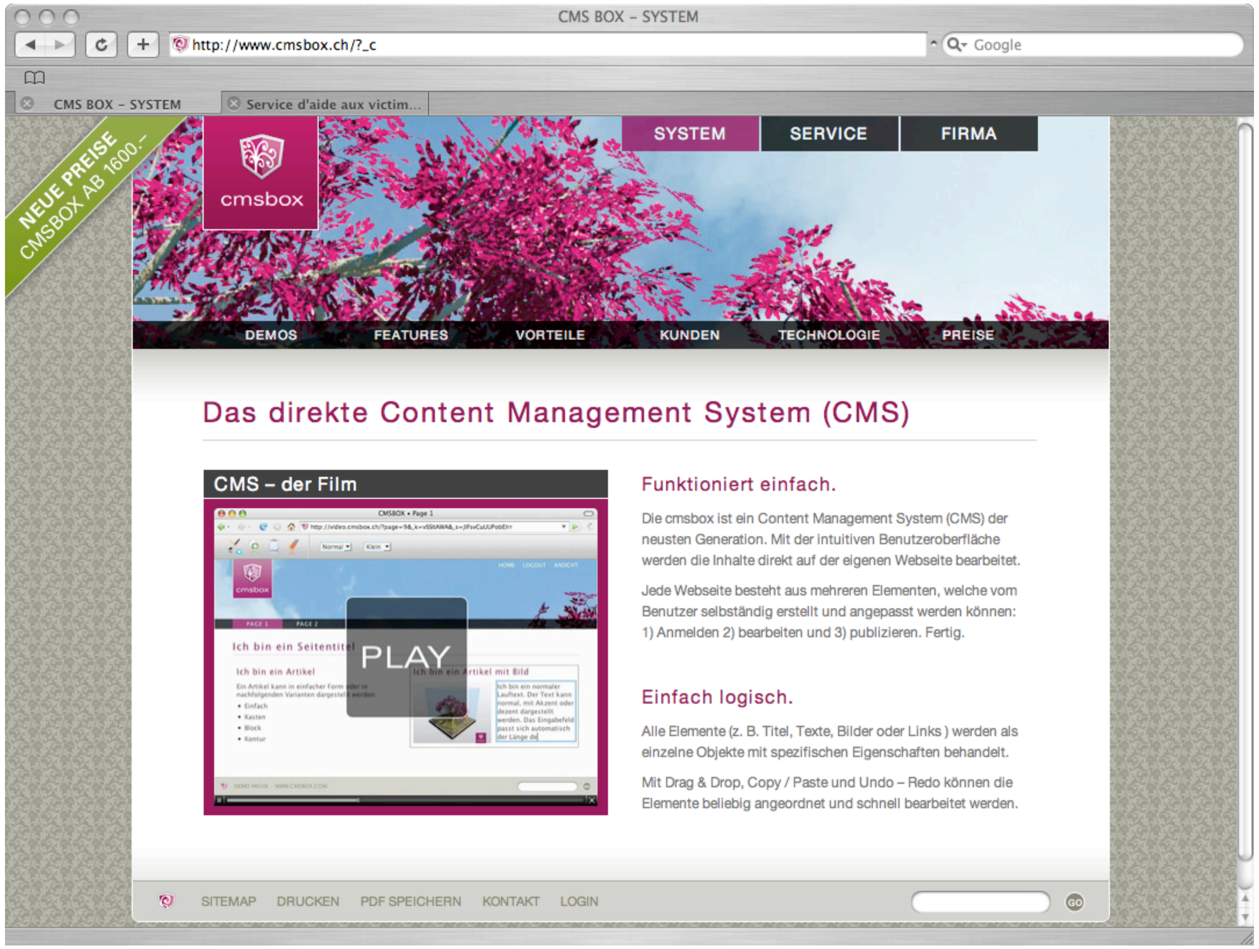
No credit card needed. Upgrade to a **paid plan** or to Dabble DB Commons at any time.

More pricing details

We also offer discounts for non-profit and academic users.

Example: CMSBox

cmsbox.ch



Das direkte Content Management System (CMS)

Funktioniert einfach.

Die cmsbox ist ein Content Management System (CMS) der neusten Generation. Mit der intuitiven Benutzeroberfläche werden die Inhalte direkt auf der eigenen Webseite bearbeitet.

Jede Webseite besteht aus mehreren Elementen, welche vom Benutzer selbständig erstellt und angepasst werden können: 1) Anmelden 2) bearbeiten und 3) publizieren. Fertig.

Einfach logisch.

Alle Elemente (z. B. Titel, Texte, Bilder oder Links) werden als einzelne Objekte mit spezifischen Eigenschaften behandelt.

Mit Drag & Drop, Copy / Paste und Undo – Redo können die Elemente beliebig angeordnet und schnell bearbeitet werden.

b UNIVERSITÄT BERN

Roadmap

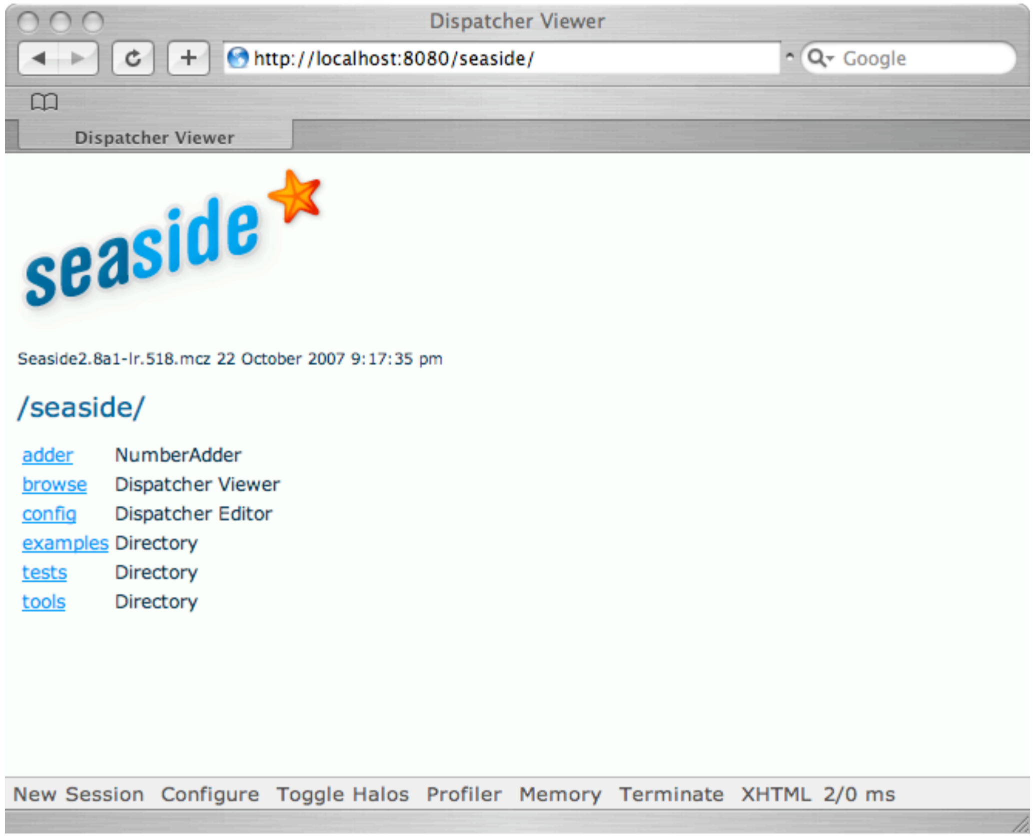
Installing Seaside

> Easy: Get from
  - http://www.iam.unibe.ch/~scg/Teaching/Smalltalk/Exercises/05Seaside/
  - For the config tool: user: admin, passwd: seaside

> Or install by hand
  - Install Seaside 2.8a via SqueakMap
  - Update using Monticello
  - WAKom startOn: 8080
  - Point browser to http://localhost:8080/seaside

Seaside

http://localhost:8080/seaside


Dispatcher Viewer
http://localhost:8080/seaside/
Google
Dispatcher Viewer
seaside

Seaside2.8a1-lr.518.mcz 22 October 2007 9:17:35 pm

/seaside/

adder	NumberAdder
browse	Dispatcher Viewer
config	Dispatcher Editor
examples	Directory
tests	Directory
tools	Directory

New Session Configure Toggle Halos Profiler Memory Terminate XHTML 2/0 ms

Roadmap

> Introduction
  - Web applications / Overview
  - Installation

> **Control Flow**

> Components

> Composition

2. Control Flow

Defining Flow

> Create a subclass of `WATask`
  - Implement the method `#go`
  - Split the method `#go` into smaller parts to ensure readability

> Tasks are a special kind of component
  - No visual representation
  - Define a logical flow `(#go)`
  - Call other components for output

Convenience Methods

- #inform: aString
- #confirm: aString
- #request: aString
- #request:label:default:

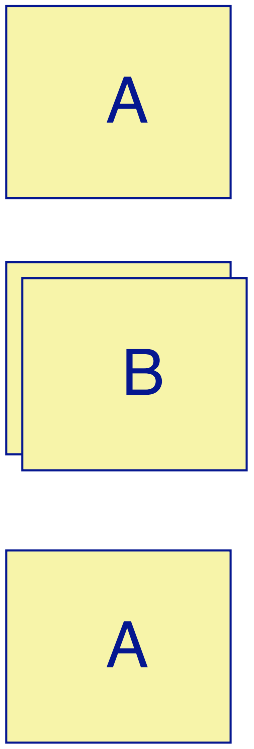
Call and Answer

> #call: aComponent
  - Transfer control to aComponent

> #answer: anObject
  - anObject will be returned from #call:
  - Receiving component will be removed

Call and Answer

Client



Server

```
A>>go
    x := self call: B
    x asString.


B>>go
    …
    self answer: 77.


A>>go
    x := self call: B.
    x astring.
              -> 77
```

Transactions

> Sometimes it is required to prevent the user from going back within a flow
> Calling #isolate: treats the flow defined in the block as a transaction
> Users are able to move back and forth within the transaction, but once completed, they cannot go back anymore

Example for #isolate:

```
self isolate: [
   self doShopping.
   self collectPaymentInfo].
Self showConfirmation.
```

Roadmap

> Introduction
  — Web applications / Overview
  — Installation
> Control Flow
> **Components**
> Composition

3. Components

> Rendering
  - XHTML
  - CSS
> Callbacks
  - Anchor
  - Form
> Customization

Components

- Components are the *Views* (and *Controllers*) of Seaside applications.
- Components keep their state (model and state of user-interface) in *instance-variables.*
- Components define the *visual appearance* and handle *user interactions*

Building Components

- Components are created by subclassing `WAComponent`
- Add instance-variables to hold your model and user-interface state
- Put view related methods in a category called *rendering*
- Put controller related methods into method categories named *accessing, actions, private*

Rendering

> XHTML is built programmatically.
> This process is called rendering.

> Create a method called #renderContentOn:

```
SomeComponent>>renderContentOn: html
   html text: ‘Hello World’
```

Text Rendering

> Render a string:

```
html text: ‘My Text’
```

> Render an un-escaped string:

```
html html: ‘<foo>Zork</foo>’
```

> Render any object (using double dispatch):

```
html render: 1
```

Canvas and Brushes

> `html` parameter is instance of `WARenderingCanvas`
  - Basic html output
  - Render logic

> Canvas provides brushes
  - For rendering html tags

Basic Brushes

> Render a new line `<br />`:

```
html break.
```

> Render a horizontal Rule `<hr />`:

```
html horizontalRule.
```

> Render a non-breaking space `&nbsp;`:

```
html space.
```

Using Brushes

1. Ask the canvas for a div brush

```
html div.
```

2. Configure the brush, e.g. set attributes

```
html div class: ‘beautiful’.
```

3. Render the contents of the tag-brush:

```
html div
  class: ‘beautiful’;
  with: ‘Hello World’.
```

Painting with Brushes

Seaside

```
html div


html div
  class: ‘beautiful’


html div
  class: ‘beautiful’;
  with: ‘Hello World’.
```

XHTML

```
<div></div>


<div class=“beautiful”>
</div>


<div class=“beautiful”>
Hello World
</div>
```

Nesting Brushes

> Render a text in **bold**:

```
html strong with: ‘My Text’.
```

> Render a text in *italic*

```
html emphasis with: ‘My Text’.
```

> Render a text in **bold** and *italic*:

```
html strong with: [
  html emphasis with: ‘My Text’].
```

Nesting Brushes

> To nest brushes use the message `#with:`.
> Always send `#with:` as the *last message* in the configuration cascade.
> The argument of `#with:` is rendered using double-dispatch, therefore any object can be passed as an argument.
> To nest tags, pass a block that renders the elements to nest.

Nesting Brushes

> Render nested divs:

```
html div id: ‘frame’; with: [
   html div id: ‘contents’; with: …
   html div id: ‘sidebar’; with: … ].
```

> Render a list:

```
html orderedList with: [
   html listItem with: …
   html listItem with: … ].
```

Rendering Pitfalls I

> Don’t change the state of the application while rendering, unless you have a really good reason to do so.
>
> Rendering is a *read-only* phase.
>
> Don’t put all your rendering code into a single method. Split it into small parts and chose a method name following the pattern `#render*On:`

Rendering Pitfalls II

- Rendering is a *read-ony* phase.
  - Don’t send `#renderContentOn:` from your own code, use `#render:` instead.
  - Don’t send `#call:` and `#answer:` while rendering
- Always use `#with:` as the last message in the configuration cascase of your brush

Anchor Callback

UNIVERSITÄT BERN

> Ask the rendering canvas for an anchor and configure it with a callback-block:

```
html anchor
   callback: [self someAction]
   with: 'Some Action'].
```

> The callback-block is cached and will be executed later.

Anchor Example

```
WACounter>>renderContentOn: html
   html heading
        level: 1;
        with: self count.
   html anchor
        callback: [self increase];
        with: ‘++’.
   html space.
   html anchor
        callback: [self decrease];
        with: ‘--’.
```

Forms

> Render a form around your form elements:

```
html form: [ … ]
```

> Put the Form elements inside the form:

```
html form: [
   html textInput
     value: text;
     callback: [:value | text := value].
   html submitButton ].
```

More Brushes with Callbacks..

> Text Input / Text Area
> Submit Button
> Check-Box
> Radio Group
> Select List
> File-Upload

Have a look at the tests!

Register new Component / Task

> Create method `#canBeRoot` returning true on class side

> Register using Seaside configuration interface.

> Or call `#registerAsApplication:` in the class-side `#initialize`

Roadmap

> Introduction
  - Web applications / Overview
  - Installation
> Control Flow
> Components
> **Composition**

4. Compositon

> Backtracking

> Subcomponents

> Widgets

Backtracking State

> Seaside does not backtrack state by default.

> Often it is not obvious whether an object should be backtracked or not. Mostly this has to be decided by the developer on a per-object basis.

> Any object can be declared to be backtracked.

Shopping Cart Problem

- Online Bookstore (without backtracking)
  - When using the back-button, usually the items should not be removed from the cart; just resume browsing from the old location.
- Flight Reservation System (with backtracking)
  - When using the back-button, usually you want to check other flights, this means the selected flight should be removed.

Register Object

- Implement method `#states` that returns an Array that contains your object.
- This will backtrack the *instance-variables* of the objects, not the objects themselves

```
SomeComponent>>#states
    ^ Array with: model.
```

Subcomponents

> It is common for a component to display instances of other components.
> Components can be nested into each other using the composite pattern.
> A subcomponent is displayed using the method `#render:` on the canvas.

Initialize Children

> Subcomponents are usually stored within instance variables of the parent component.
> Subcomponents are commonly created lazily or as part of the components `#initialize` method.

```
SomeComponent>>initialize
   super initialize.
   counter := WACounter new.
```

Enable Children

> Parent Components *must* implement a `#children` method returning a collection of subcomponents that they *might* display.

> If you fail to specify `#children` correctly, Seaside will raise an exception.

```
SomeComponent>>children
    ^ Array with: counter
```

Render Children

> Children are rendered by sending the message `#render:` to the rendering canvas.

> Never directly send `#renderContentOn:` to the subcomponent.

```
SomeComponent>>renderContentOn: html
   html heading level: 1; with: 'My Counter'.
   html render: counter.
```

Widgets

> Components can be reused in different contexts within different applications.

> Seaside is shipped with a small collection of widgets ready to use.

> Load and use widgets that have been developed by the Seaside community.

> Write your own widgets that exactly fit your needs

Widgets: Examples

- Batched List
- Tab Panel
- Calendar

Have a look at the classes in
Seaside-Components-Widgets

Custom Widgets

- Create a new component.
- Add methods to specify domain-model, subcomponents, properties…
- Assign CSS names/classes to make it skinnable with css style-sheet.
  - Implement method `#style` to return CSS for component
- Write tests and small example applications.

There is more..

> Development Tools
  - Demo in the Exercise Session (Halo, Configuration…)
  - Debugging: Next Lecture

> AJAX and script.aculo.us

> Persistency (Databases)

Literature

> HPI Seaside Tutorial:
  — http://www.swa.hpi.uni-potsdam.de/seaside/tutorial

> Ducasse, Lienhard, Renggli: **Seaside: A Flexible Environment for Building Dynamic Web Applications** (IEEE Software, vol. 24 no.5)

> Ducasse, Lienhard, Renggli: **Seaside, a Multiple Control Flow Web Application Framework** (Proceedings ISC04)

> …. More at Seaside.st

License

> http://creativecommons.org/licenses/by-sa/3.0/