Reef getting started tutorial - Part 1

Reef is a framework to build dynamic components. As a complement to Seaside, Reef does not replace it, instead, it makes easier the use of AJAX/Javascript inside it.

In this tutorial, we are going to create a dynamic search of classes, to show basic AJAX interaction.

Installation

You need a Pharo 1.1 image (better a PharoDev, go to http://pharo-project.org to download one), with Seaside 3.0 installed on it, then do:

```
Gofer it
    squeaksource: 'Reef';
    package: 'Reef';
    load.
```

...and that's all, currently no Metacello configuration is required, since Reef is just one package, but this can change in the future.

Creating a search component

To create a form with Reef, **you need to subclass REForm (the Reef component for make forms)**

```
REForm subclass: #RTSearchPart
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'ReefTutorial-View'
```

Then, you need to override **#initialize**, to add children components to your container widget ("Container widgets" are the widgets that allow children inside, other provided are: **REContainer**, **REPanel**, **REGrid**, etc.)

```
initializeContents
    self add: 'Search: '.
    self add: RETextField new
```

And that's your first "part component" (I usually call reef components"parts", as a convention to make explicit the fact that a Reef component is just a "part" of a Seaside component)

So, to see your creation, **you will need a Seaside component:**

```
WAComponent subclass: #RTWebApplication
    instanceVariableNames: 'searchComponent'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'ReefTutorial-View'
```

```
initialize
    super initialize.
    searchComponent := RTSearchPart new asComponent.
```

```
children
    ^Array with: searchComponent
```

```
renderContentOn: html
    html render: searchComponent.
```

The “magic” here is the **#asComponent** message call. This message says to any Reef component to be wrapped into a Seaside component. This component can now be added to your Seaside application as any other component.

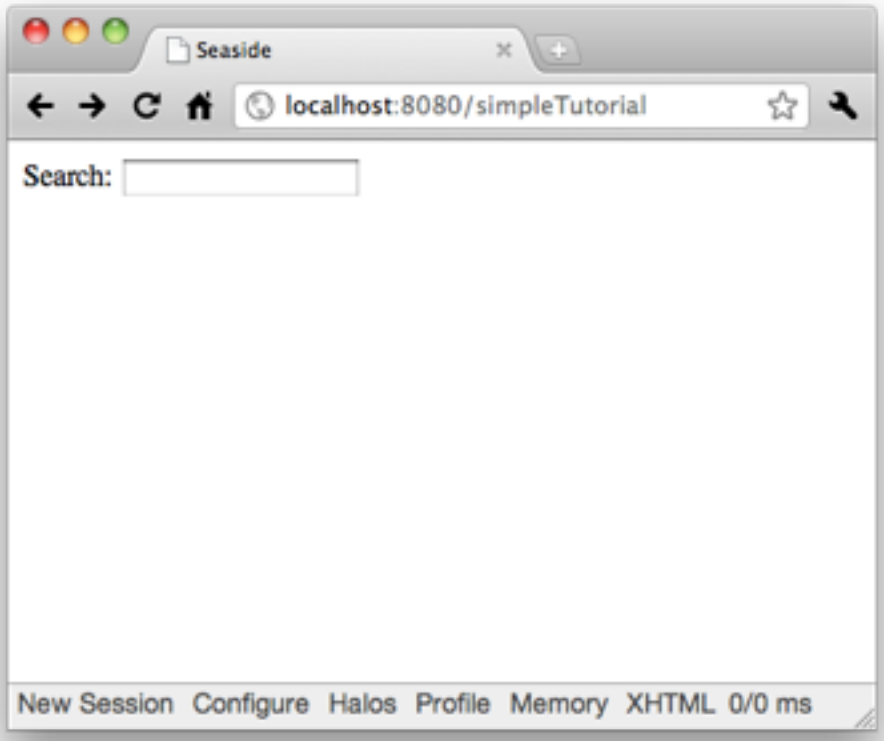
So, now we need to **register our application,** as a Reef application, execute in a workspace:

```
REApplication
    registerAsApplication: 'simpleTutorial'
    root: RTWebApplication.
```

So, why we are using this way to register, and not the standard way? Well, because a Reef application needs to be configured with some elements:
1) We need some jQuery libraries
2) We need a **div tag** always present in our browser (a “dispatcher area” to process AJAX requests)

REApplication>>#registerAsApplication:root: does that for you, but you could do this by yourself and use the common application registering mechanism.

So, executing your application in http://localhost:8080/simpleTutorial, we get:



Ok, no so great at the moment, but we are going to add more functionalities.

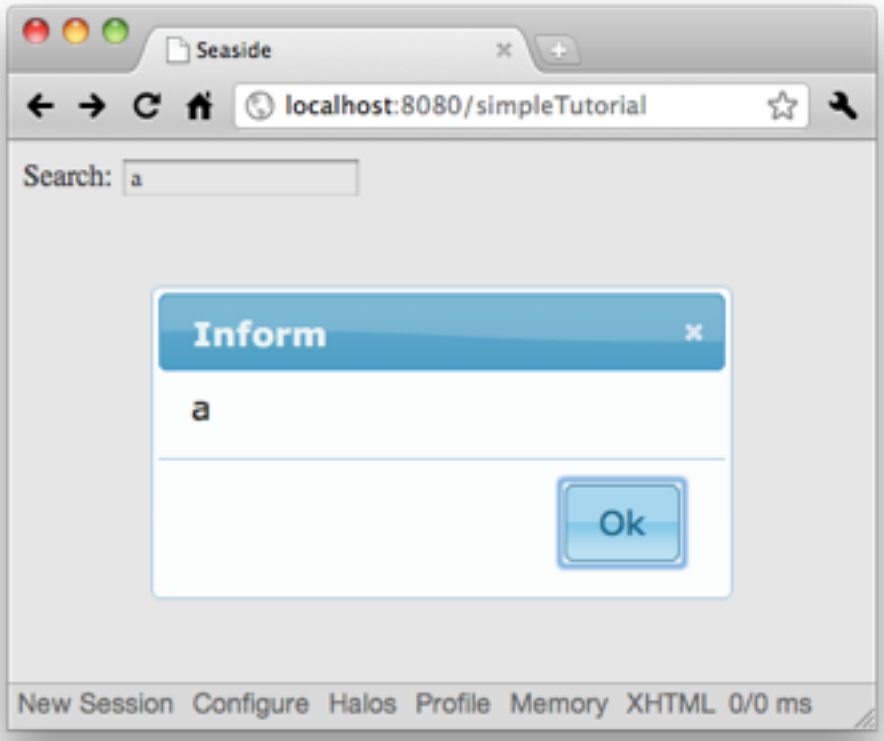
Right now, our Reef component doesn't do anything (just render a form with a text field), but this was just the first step, now we are going to add behavior to our text field widget.

```
initializeContents
    self add: 'Search: '.
    self add: (RETextField new
        onKeyPress: [ :me | me triggerThenDo: [ self inform: text ] ];
        callback: [ :v | text := v ]).
```

Changes introduced:

1) We added a **#callback:** message. The **#callback:** looks like a standard seaside brush... and in fact is the same.
2) We added a **#onKeyPress:** message. This is different than Seaside. In plain Seaside, you need to send a Javascript string to this kind of messages (tag events). **In Reef, that's not what you do. Instead, you use a block with Smalltalk code (at least in most cases).** In this case, the **#onKeyPress:** block is getting the text field as a parameter (it is optional), and we are saying: trigger this widget and then execute another action.

Let's test it!



So, this is better, but still far from our search component.

Adding a “results panel”

Now we want to add a panel to show the classes searched. For this we are going to modify our **RESearchPart** again.

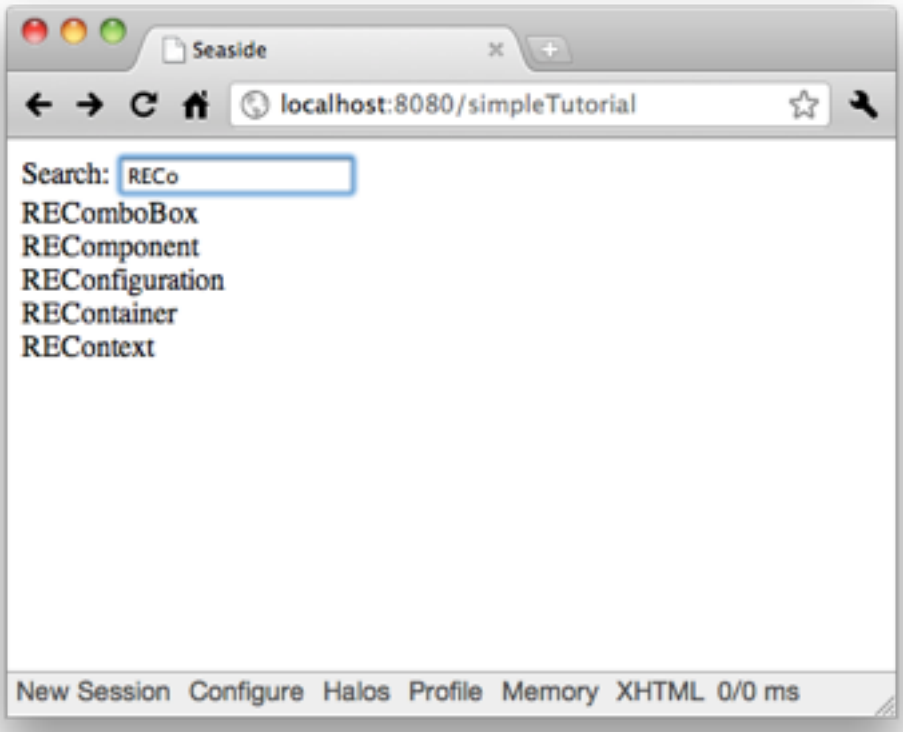
```
initializeContents
    resultsPanel := REPanel new.
    self add: 'Search: '.
    self add: (RETextField new
        onKeyPress: [ :me | me triggerThenDo: [ self search ] ];
        callback: [ :v | text := v ]).
    self add: resultsPanel
```

```
search
    resultsPanel removeAll.
    text ifNotEmpty: [
        resultsPanel
            addAll: (Smalltalk classNames
                select: [ :each | each beginsWith: text ]
                thenCollect: [ :each |
                    REPanel new
                        add: each;
                        yourself ]);
            refresh ]
```

What changes we introduced?

1) we added a panel (resultPanel)
2) then, we modified **#onKeyPress:** to call **#search** instead showing a dialog.
3) **#search** method does a refresh of the panel by doing:
    a. remove all children (if any)
    b. add all class names with a name starting with “text”, as a new panel (we need a panel because we want a name per line...)
    c. send a **#refresh** message, who actually performs the refresh of the panel content.

This is what we get now:



And that’s all for now... this a a very simple component. In future tutorial parts, we are going to show more complex elements.