Client-Server interaction in Reef

(Or callbacks revisited/redefined)

One of the most important features in Seaside is it’s natural flow, achieved through the use of closures, just as the way of solve the navigation in Smalltalk. One of the problems on using a client-side library, such as jQuery or any Javascript library, is the loose of this natural feeling.
In Reef, we try to keep this natural flow, as much as we can, by hiding most of the javascript complexity, and using smalltalk closures also for javascript behavior. For that, we have redefined the event behavior to match a little more with what a smalltalk programmer can expect.

Three different behaviors

There are three kind of behaviors possible in the interaction between the page in client and the Seaside server:

1. The client application needs to access the server to make something, and then update the status of the page, without rendering a new page. As we think this kind of interaction is the hardest to manage, and the most common used, **this is the default Reef behavior**.
2. The client application needs to call server with some functionality, and then renders a new page. This is the “normal behavior” of a web application, and in Reef we support them as a way to “get back” to normal flow, while inside a Reef flow.
3. The page needs to change the DOM (for instance, to hover current selected row), without going to server at all. This is, for us, the most difficult to solve, because the DOM manipulation needs to be a javascript sentence.

This three different behaviors are defining the ways an application can interact with the user, creating different results, but just one final feeling: we are in a “desktop” application. We are going to explain how Reef manages this three approaches.

Updating the server, then update the client.

As we said before, we think this is the most common behavior you can find in a web-application, and for that reason we are focusing in this mode.
How does it work?
Well, let’s suppose you have a validation in a text field, who needs to be validated on blur event (once you lost focus), you have something like this:

```
...
self add: (RETextField new
        onBlur: [ :me |
                me triggerThenDo: [
                        self validate ifFalse: [ self error: ‘Error validating’ ] ] ];
        on: #value of: self)
...
```

As you can see, the flow is more or less like it would be in any usual Smalltalk application, with just one minor difference: you need to tell the client that you are going to use the value just typed in this field, and just then we can **for real** interact with the client side value.

Using this kind of callback is the **default** behavior in any Reef component, with the unique exception of the argument of #callback: in form elements (those who will become in a FORM element in HTML, like an INPUT, TEXTAREA, etc.).

Update the server, then render a new page
This functionality is useful after pressing a button, or after doing some operation who we decide needs a new page. What it does is skip the client/server AJAX interaction and render a full new page (it works just like a common anchor or button press in plain HTML). To tell Reef that the result of a callback should be executed in another page, we use the protocol **#asReefPageCallback**. See next example:

```
...
add: (REButton new
        label: 'Login';
        callback: [ self login ] asReefPageCallback)
...
```

In this case, pressing the button acts just like pressing a regular button in Seaside (well... not **exactly** equal, but act like that in almost any case).

Not updating the server, just the client
This third case useful to make some client side validations, or highlighting something on hover. This callback execute it's content in render time, and prepare the page with the resulting javascript, so it can act in client side.
To tell Reef that it should execute the callback in client side, we use the protocol: **#asReefClientCallback**. See the example:

```
...
self add: (REPanel new
        class: 'panel';
        onMouseMove: [ :me | me asClient addClass: 'hover' ] asReefClientCallback;
        onMouseOut: [ :me | me asClient removeClass: 'hover' ] asReefClientCallback;
        add: 'Panel with hover';
        yourself).
...
```

As you can see, we introduced the protocol **#asClient**, which answer a representation in JavaScript, and you can then send messages of the jQuery package.

Note: Yes, this is my worst creation, I still can not found a better way to do this... but it works, almost properly :)

Hybrids

There is still one scenario to deal: That in which we want to do something and in some cases the opposite (For example, if we want to validate and if succeed render a new page, and if fails stay in current page, and show a message). To allow us execute different things inside a flow, there are the protocols: **#inPage:**, **#inClient:**, both allow us to perform different actions to change the render logic. See the example:

```
...
self add: (REButton new
        label: ‘Login’;
        callback: [
                self triggerThenDo: [
                        self login
                                ifTrue: [ self inPage: [ self show: aComponent ] ]
                                ifFalse: [ self inClient: [ self asClient addClass: ‘error’ ] ] ] ].
...
```

In this case, a new page (with aComponent contents) will be rendered if the login is successful, and an error class will be added to form if it doesn’t.

Vocabulary

This is what we show in this post:

[...]	A block in Reef is a callback who leads to an interaction between client and server, without render a new page. Exception: callbacks on form elements.
[...] asReefPageCallback	Contents of block are executed, but all rendering is do it in a new page.
[...] asReefClientCallback	Contents of block are executed in rendering phase, and the result must be a JQuery Stream or a String.
[:me I ...]	All blocks can receive the owner as an argument (but it is not mandatory)
#triggerThenDo:	Triggers the form element (if applied to a form triggers full form)
#asClient	jQuery object representation of reef component.
#inPage:	Renders the block contents in a new page
#inClient:	Renders the block, in same page as we are now. It works just like #asReefClientCallback.