RWT

Rails Web Toolkit

Author: Sérgio M. Brant

Revision: 07/19/09

Introduction

Explaining through tests

The idea here is to explain the rwt API by explaining the rwt Rspec tests. I'll start with the most basic rwt class, the Component class.

Rwt has a buffer that stores the javascript code being generated during the processing of rwt views. The buffer can be cleaned using the *Rwt.clear* method. Normally the *rwt_render* method will clear the buffer, process the view being rendered and send the javascript code generated in the buffer to execution in the client browser.

Lets take a look at our first test of our *component_spec.rb* test suite:

```
 it "should generate javascript in the buffer" do
   component(:a_parameter=>'a_value')
   Rwt.code.should include("v:") # all components has a unique identification stored in the 'v' parameter
   Rwt.code.should include("a_parameter:'a_value'")
# puts Rwt.code
 end
```

If we uncomment the puts line we can see what is being generated in the Rwt.code buffer:

```
var v1=new Ext.Component({a_parameter:'a_value',v:'v1'});
```

This is an instatiation of a javascript component (Ext.Component) with the *a_parameter* parameter set to *'a_value'*. The *component* method of the Rwt module instantiates a Rwt::Component class that generates (during its initialization) the corresponding javascript instatiation code of the ExtJs Ext.Component class. The additional *v* parameter is just an auxiliary place to store the name of the javascript variable that will point to the javascript object created at run-time.

So, whenever we call *component* in a rwt ruby view we will get new code generated in the rwt buffer that later will be used by rwt_render sending it to be executed in the client browser.

Rwt components are inherits from Rwt::Component. Any rwt component can have default non-hashed parameters defined by the *init_default_par(non_hash)* overridable method. Component itself has the *text* parameter as the only non-hashed default parameter:

```
 it "should accept the :text parameter as the first non keyed value" do
   component('First Component')
   Rwt.code.should include("text:'First Component'")
   component(:a_parameter=>'a_value',:text=>'Second Component') # can use keyed form too
   Rwt.code.should include("text:'Second Component'")
#   puts Rwt.code
 end
```

The next test exercises the capability of constructing a hierarchy of components with rwt:

```
 it "should be able to construct a hierachy of components" do
   a_gf=component('grandfather') do |gf| # gf here only needed because of reference in son
     component('father') do |f|
       f.owner.config[:text].should == 'grandfather'
       component('son') do |s|
         s.owner.config[:text].should == 'father'
         s.on("create") do
           Rwt << "#{gf}.show();"
         end
       end
     end
```

```
   end
#    puts Rwt.code
#    puts a_gf.vid

   Rwt.code.should include("#{a_gf}.show()")
 end
```

Wich led to the folowing javascript code:

```
var v4=new Ext.Component({v:'v4',text:'son'});
v4.on('create',function(){v2.show();});
var v3=new Ext.Component({v:'v3',items:[v4],text:'father'});
var v2=new Ext.Component({v:'v2',items:[v3],text:'grandfather'});
```

As you can see, the creation is done in reverse order to avoid errors in javascript execution.

The << method of Rwt inserts text directly in the code buffer. In the case showed here this is done in the moment of event code creation.