ChiliProject - Bug # 816: Liquid legacy layer does not handle macros which generate HTML

Status:	Closed	**Priority:**	Normal
Author:	Andreas Schuh	**Category:**	Text formatting
Created:	2012-01-05	**Assignee:**	
Updated:	2012-01-05	**Due date:**	
Remote issue URL:			
Affected version:	3.0.0		
Description:	The Liquid legacy layer does not register the macros as Liquid tags using the option html => true which would be required for macros which output HTML instead of Textile formatted text.		

History

2012-01-05 09:30 am - Holger Just

- Target version deleted (3.0.0)

- Category changed from Wikis to Text formatting

- Affected version set to 3.0.0

- Status changed from Open to Needs more information

Why would you think that?

The macro compatibility layer is in source:/lib/redmine/wiki_formatting/macros.rb. There it registers the macro wrapped as a tag with @:html => true@. See https://github.com/chiliproject/chiliproject/blob/v3.0.0beta1/lib/redmine/wiki_formatting/macros.rb#L60 for the implementation in the 3.0.0beta1.

2012-01-05 11:12 am - Andreas Schuh

Yes, I just discovered this myself and wanted to update this issue. Sorry for the false alarm.

Thing is that I am having problems with a macro which works just fine with Redmine 1.3.0, but not with ChiliProject 3.0.0beta1 :(

2012-01-05 11:36 am - Holger Just

If you provide a bit more information, we might be able to help.

Where can we find the source code of the macro? How do you call it? What output do you expect? What is the actual output?

2012-01-05 12:51 pm - Andreas Schuh

That would be great. Yet, the sources are nowhere besides my local working copy b/c the original plugin wasn't written by me and as it is Subversion, I didn't bother cloning it yet. Just started today with it.

In particular, I am working on the "Download button plugin":http://www.redmine.org/plugins/download written by Andriy to make it fit into ChiliProject 3. Further, I thought a Wiki macro which enables one to place such button anywhere in a Wiki page (including the sidebar using, for example, Andriy's "Sidebar content plugin":http://www.redmine.org/plugins/sidebar) might be useful. I noticed such functionality on the new SourceForge beta Wiki.

As for the Liquid tags, I need a class derived from ChiliProject::Liquid::Tags::Tag and am in a different scope as where the macro code was executed, I had to, for example, replace all calls to render(), content_tag(),... all those ActiveView helpers.

How can I implement a Liquid tag which makes use of ActiveView ? I.e., as it was possible with the Redmine WikiFormatter macros.

The currently remaining issue is the error: *Liquid error: undefined method `url_for' for nil:NilClass*. Note that this method is called in the DownloadHelper#download_button method, see "here":http://subversion.andriylesyuk.com/redmine-download/app/helpers/download_helper.rb.

Currently, I am doing the following (note that I modified the @download_button()@ method to not use @render()@):

<pre><code class="ruby">
```
module DownloadButtonMacroImpl
    def execute(args)
        # this version of extract_macro_options() has no problem with URL's
      # which include = signs as option value
      fixed_extract_macro_options = lambda { |*keys|
        options = {}
        while args.last.to_s.strip =~ %r{^([^=]+)\=(.+)$} && keys.include?($1.downcase.to_sym)
          options[$1.downcase.to_sym] = $2
          args.pop
        end
        return [args, options]
      }
      # parse macro options and do some other stuff
      # [...]
      # instantiate new (temporary) download button
      download = DownloadButton.new(attributes)
      raise "Failed to instantiate DownloadButton" unless download
      # render download button - could not use content_tag() here any more
      out = <<-HTML
       <div class="download-button" style="#{outer_style}">
        #{download_button(project, download, :style => inner_style)}
       </div>
       HTML
      out
    end
end

begin

    class DownloadButtonTag < ChiliProject::Liquid::Tags::Tag
        include ActionView::Helpers::TranslationHelper # t(ranslate)
        include ActionView::Helpers::UrlHelper        # url_for
        include ERB::Util                             # h(tml_escape)
        include DownloadHelper                        # download_button
        include DownloadButtonMacroImpl               # implementation

        def initialize(tag_name, markup, tokens)
          tag_args = markup.strip.gsub(/^[("']|["')]$/, '')
          if tag_args.present?
            @args = tag_args.split(',')
          else
            @args = []
          end
          super
        end

        def render(context)
          @project = Project.find(context['project'].identifier ) if context['project'].present?
          execute(@args)
        end
```

```
    end

    ChiliProject::Liquid::Tags::register_tag('download_button', DownloadButtonTag, :html => true)

rescue

  class DownloadButtonMacro
    include ActionView::Helpers::TranslationHelper # t(ranslate)
    include ActionView::Helpers::UrlHelper         # url_for
    include ERB::Util                              # h(tml_escape)
    include DownloadHelper                         # download_button
    include DownloadButtonMacroImpl                # implementation

    def initialize(project)
      @project = project
    end
  end

  Redmine::WikiFormatting::Macros.register do
    desc "Inserts Download button in Wiki pages"
    macro :download_button do |obj, args|
      impl = DownloadButtonMacro.new(@project)
      impl.execute(args)
    end
  end

end
```

BTW All the code for the macro lives at the moment in the init.rb file of the plugin. Where would you usually put such code? I am new to both Ruby and Rails...

On a side note, I had to replace the use of @l@ (I guessed an alias for @localize@, but actually it is used for translation, by @t@, the alias for @translate@. Can you explain to me what the @l@ alias as in @l(:locale_download)@ stands for?!? See http://subversion.andriylesyuk.com/redmine-download/app/helpers/download_helper.rb

2012-01-05 07:09 pm - Andreas Schuh

- Status changed from Needs more information to Closed

Eventually, I figured it all out.

For everyone who may end up reading the above long comment in the future, the solution was basically to put all the ActiveView related rendering code into a .rhtml/.html.erb file and then to render this partial using the :view object of the context, i.e.,

```
def render(context)
  # [...]
  context.registers[:view].render :partial "download/tag", :locals => { [...] }
end
```

In my particular case, the @"download/tag"@ partial view itself calls the @DownloadHelper#download_button()@ method, which in turn renders the partial @"download/button"@, the partial which is shared among the sidebar button as known from the version 0.4.5 of the Download button plugin and the just implemented ChiliProject Liquid tag or Redmine Wiki formatting macro. I will push this enhancement of the plugin upstream.

2012-01-05 07:17 pm - Andreas Schuh

Just a correction, the current version of the Download button plugin is 0.0.2, not 0.4.5... :S