Documentation for CTS-Admin-Tool

Sascha Ludwig
sasludwi@uos.de

April 22, 2015

Contents

1 CTS-Iterator

1.1 Copy original files and init Java Eclipse Project

- CTS Java-Iterator can be found here: `http://ctstest.informatik.uni-leipzig.de/`
- download the whole zip-File `https://bitbucket.org/jtiepmar/ctsiterator/downloads`
- unpack
- create new Java-Project in Eclipse
- copy files to the new Project Folder (rename source to src)
- in Eclipse, right click on the JDOM.jar and JDOM2.jar, "Build Path → Add"
- export as single .jar file

1.2 Installing Ruby

- for installing JRuby:
  - under Linux: see here: `https://rvm.io/`
  - some rvm commands:

```
rvm get stable
rvm list known
rvm list rubies
rvm install jruby
rvm install ruby
```

  - under Windows: `http://jruby.org/download`
- installing missing gems with "gem install GEMNAME"
- calling Java methods in JRuby:
  - require 'java'
  - import NAMEOFCLASS
  - Java classes could now be used as in Java

2 Tomcat7 and CTS Installation

- under Windows use XAMPP
- install Oracle Java 8 on Ubuntu

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer oracle-java8-set-default
```

- installing Tomcat7 on Ubuntu System (there are errors with Tomcat8 though)

– **IMPORTANT: There must not be spaces in any paths !**

```
sudo apt-get install tomcat7 tomcat7-user
sudo apt-get install tomcat7-docs tomcat7-admin tomcat7-examples
```

– on Ubuntu, some directories are missing (ubuntu bug?!)

```
sudo mkdir /usr/share/tomcat7/common
sudo mkdir /usr/share/tomcat7/common/classes
sudo mkdir /usr/share/tomcat7/shared
sudo mkdir /usr/share/tomcat7/shared/classes
sudo mkdir /usr/share/tomcat7/server
sudo mkdir /usr/share/tomcat7/server/classes
```

- create a new, private Tomcat7 instance:

```
tomcat7-instance-create ctsadminsascha
```

– change ports; for config see: conf/server.xml

– add Oracle Java JDK in /bin/setenv.sh: JDK_DIRS="/usr/lib/jvm/java-8-oracle"

– see subfolder /logs for errors

– starting: /bin/startup.sh

– shutting down: /bin/shutdown.sh

– installing CTS-Iterator stuff:

– have a look at `urncts.de`

– start TomCat server

– copy cts.war to tomcat's "webapps" folder

– wait some time, maybe 15 seconds

– test successful installation by calling
`http://localhost:8080/cts/cts/?request=GetCapabilities`

– should return something like:
`<GetCapabilities><request>GetCapabilities</request><reply/></GetCapabilities>`

– configure MySQL access in /webapps/cts/WEB-INF/lib/conf.properties

– call inside /webapps/cts/WEB-INF/lib/ : java -jar Datapreparation.jar

– wait for some time for the program to finish

– check again by calling `http://localhost:8080/cts/cts/?request=GetCapabilities`

3 Ruby and Sinatra

installing required gems:

```
rvm use jruby
gem install warbler sinatra
```

or

```
rvm use jruby
bundle install
```

3.1 Ruby Sinatra Application to WAR-File

How to create a .war-file from the Ruby-Code

- Gemfile is needed (should be standard anyway)
- config.ru is needed for rake stuff
- call `jruby -S warble` in same directory where the ruby files are
- copy war-file to webapps folder
- it could not be used via the tomcat server (on errors, restart tomcat server)

Starting the Sinatra server without tomcat for development

- the Sinatra server is started by calling `rackup` in the root directory (= the directory were the config.ru is located)
- by default, the server binds to localhost:9292

4 How to create a new entry in the menu bar

- To create a new functionality that is not associated with an specific CTS-instance, the menu bar (= dark menu bar on top) is the right place
- to create a new tab there, 3 files have to be modified and 1 has the be created
  - **/views/main.erb** : [modify] this file is the main layout. This includes the menu bar (top), the side bar (left) and the cts-action-bar (beneath menu bar) but not the actual content
  - **cts_admin_root.rb** : [modify] this file handels all routes (= url calls) and the program logic. It also manages the connection to the database and OS stuff like file operations
  - **/views/admin_area.erb** : [create] this file will render the content specific to the new action
  - **/public/js/cts_admin.js** : [modify] this file handles the Javascript stuff, e.g. click on buttons and AJAX requests

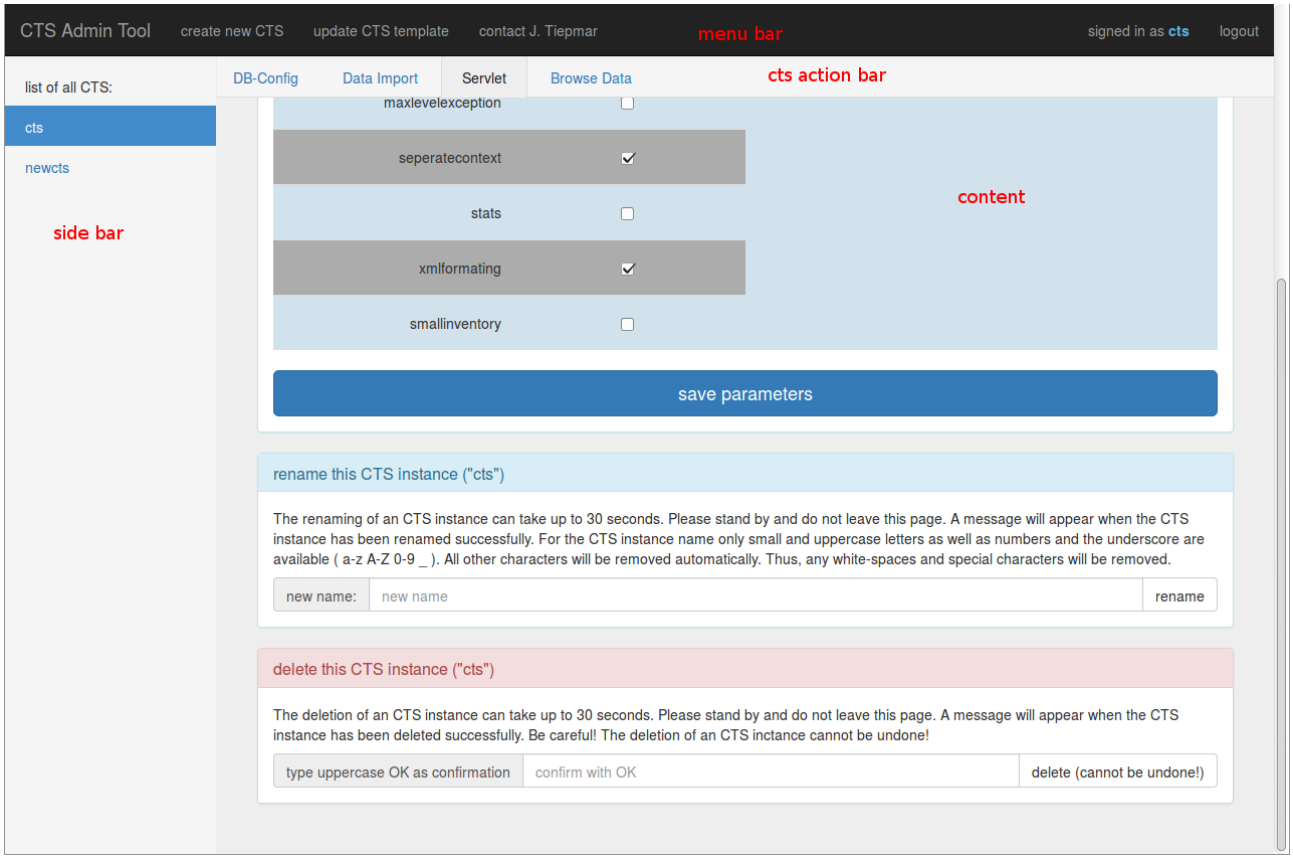

Figure 1: general layout of the CTS-Admin-Tool

4.1 modification of /views/main.erb

- the menu bar is declared at the top of the page (`<nav class="navbar navbar-default navbar-fixed-top navbar-inverse">`). search here for the unordered list `<ul class="nav navbar-nav">`
- add a new entry to this list. e.g.

```
<li><a href="<%= url("/admin_area") %>">admin area</a></li>
```

- you should now see a new tab in the menu bar that points to "/admin_area"
- if you click on it, you will be redirected to the index page since the route "admin_area" does not exist (and default routing on 404 is the index page)
- note1: the `<%= %>` will run ruby code and inject the returned string directly into the html-code
- note2: the 'url()' method will take a string and make an absolute path of it. this is necessary since tomcat runs not on the root but in a folder
- note3: always surround the ruby injections with double quotes. otherwise spaces can lead to strange html attributes

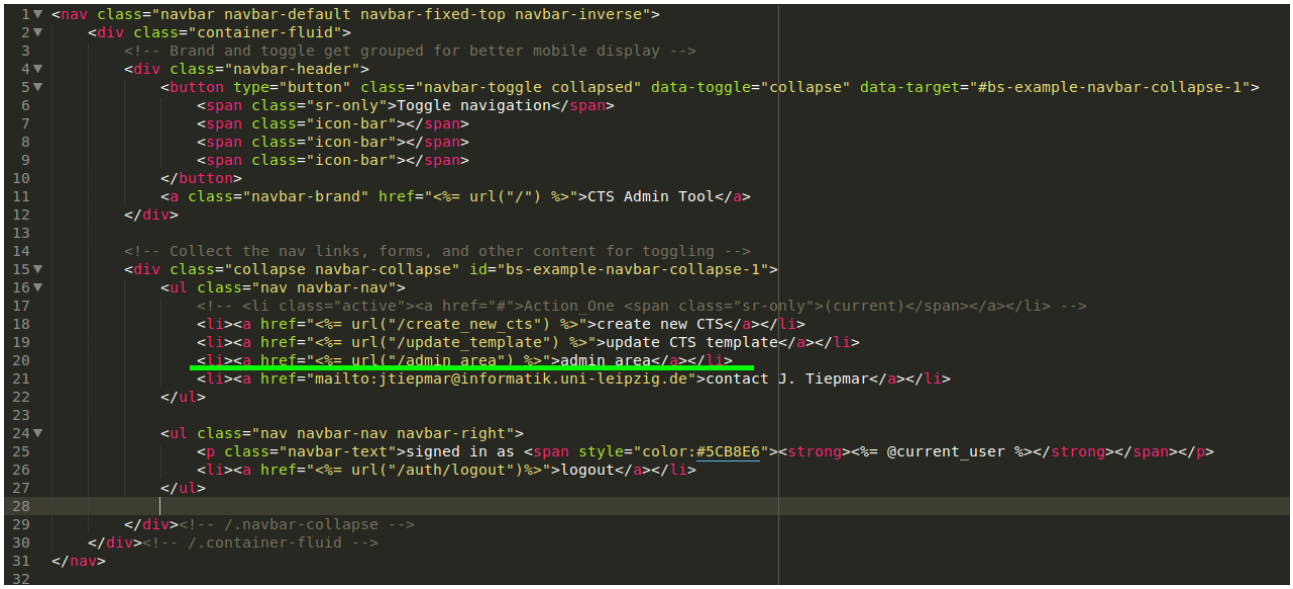

Figure 2: step 1 in /views/main.erb

4.2 modification in cts_admin_root.rb

- now we have the create a route to handle the request for "/admin_area"
- this is done by adding another get-handler inside our Sinatra class in cts_admin_root.rb

```
# shows an administration area
get '/admin_area' do
        # needs authentication
        env['warden'].authenticate!
        # specific data
        @cts_file_marker = "CTSStore.jar"
        # show stuff
        @sub_erb = :admin_area
        erb :main
end
```

- note1: if this page should be accessible without authentication, remove "env['warden'].authenticate!"
- note2: @sub_erb specifies the erb-file (default path in /views) that should be rendered in the content area
- note3: the last statement "erb :main" will return a html-string (from /views/main.erb) which will in turn be rendered inside the layout.erb (default by name convention) at the yield statement
- note4: the layout.erb only holds all css and java-script declarations as well as some meta-tags. this file is used for every request. whereas the main.erb is the second-level layout. It is mostly used but not for the login-screen.
- note5: thus, the render structure is: layout.erb → main.erb → @sub_erb (here: admin_area.erb)

```ruby
# upload a new cts file
get '/update_template' do
    # needs authentication
    env['warden'].authenticate!
    # data
    @up_ret     = nil
    # show stuff
    @sub_erb    = :cts_upload_template
    erb :main
end


# shows an administration area
get '/admin_area' do
    # needs authentication
    env['warden'].authenticate!

    # specific data
    @cts_file_marker = "CTSStore.jar"

    # show stuff
    @sub_erb = :admin_area
    erb :main
end
```

Figure 3: step 2 in cts_admin_root.rb

4.3 creation of /views/admin_area.erb

- at this point, if we click on the menu-link to the admin area an error will occur since we have not created any file that hold our content
- thus, we have to create the file **/views/admin_area.erb** and fill it with some basic content

```
<h1>Admin Area</h1>

<p>file-marker: <%= @cts_file_marker %></p>
```

- the code is basically plain HTML with some extra ruby-features (erb = embedded ruby)
- the content of the variable `@cts_file_marker` will rendered in the paragraph-tag
- variables starting with an @ are instance variables. The scope of these variables is the current get or post-handler (that we have just created above). This includes the call of the erb-templates. Thus, all instance variables defined in the get/post-handler could be accessed
- after restarting the Sinatra server (STRG+C and then `rackup`) and going to the webadmin area we will see `file-marker: CTSStore.jar`

```erb
<h1>Admin Area</h1>

<p>file-marker: <%= @cts_file_marker %></p>
```

Figure 4: step 3 in /views/admin_area.erb

4.4 optional: Javascripts

- you could add your custom scripts at the end of the /views/admin_area.erb file

- or you could write it at the public/js/cts_admin.js

5 How to create a new entry in the CTS-action-bar

- To create a new functionality that is specific for an CTS-instance, the CTS-action-bar is the right place
- to create a new tab there, 3 files have to be modified and 1 has the be created
  - **/views/main.erb** : [modify] this file is the main layout. This includes the menu bar (top), the side bar (left) and the cts-action-bar (beneath menu bar) but not the actual content
  - **cts_admin_root.rb** : [modify] this file handles all routes (= url calls) and the program logic. It also manages the connection to the database and OS stuff like file operations
  - **/views/cts_myaction.erb** : [create] this file will render the content specific to the new action
  - **/public/js/cts_admin.js** : [modify] this file handles the Javascript stuff, e.g. click on buttons and AJAX requests

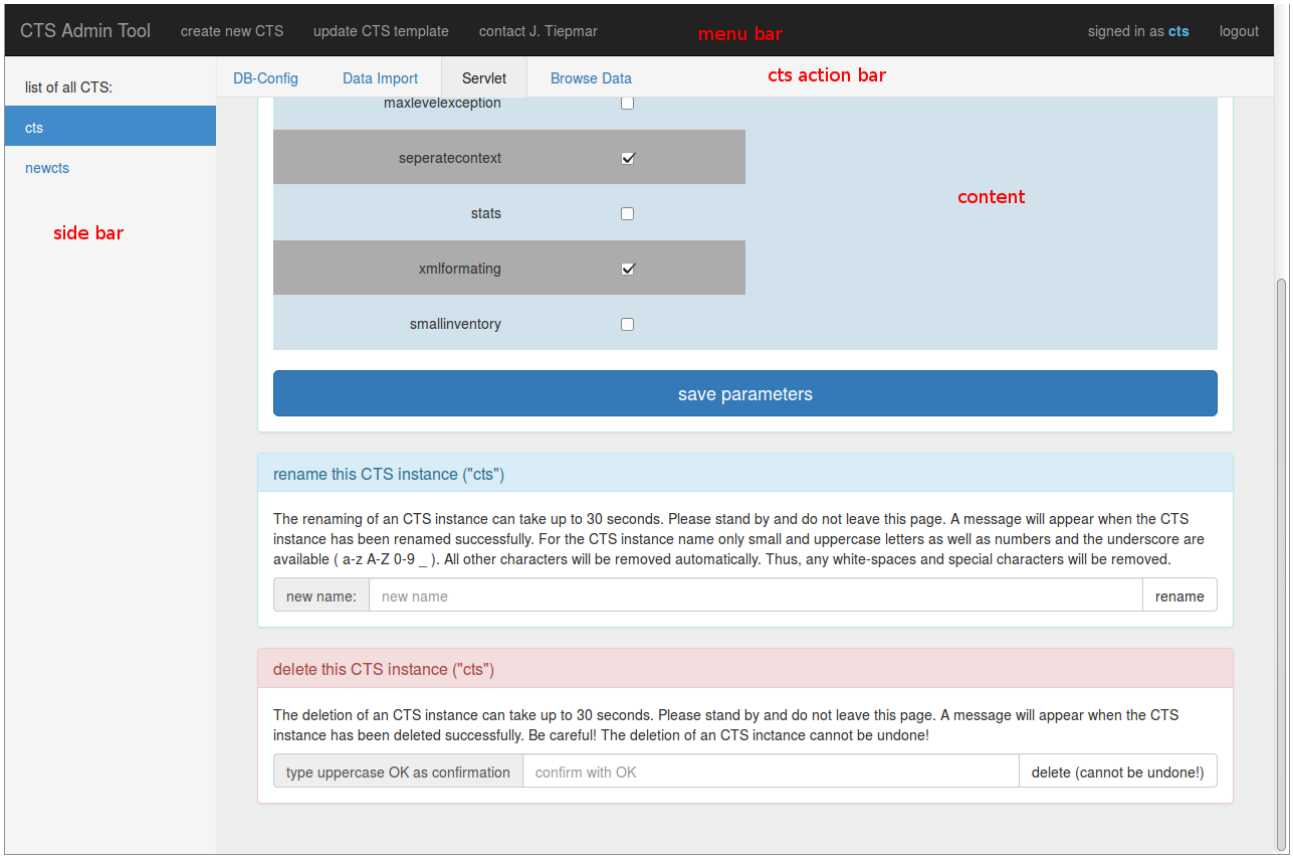

Figure 5: general layout of the CTS-Admin-Tool

5.1 modification of /views/main.erb

- go to the cts-action-bar part of the file (at the end) and find the
  `<ul class="nav nav-tabs nav-controlbar">`
- add another line

```
<li role="presentation" class=<%= "active" if @action == "myaction"%>>
<a href="<%= url("/forcts/#{@ctsname}/action/myaction") %>">Action</a>
</li>
```

- the `"active" if @action == "myaction"` ensures that the tab is marked as active when this action is selected (nicer design)
- the url for cts actions is built as follows: `/forcts/CTSNAME/action/ACTIONNAME`
- the action name should not contain spaces or any special characters

```
<!-- Sidebar and  main content -->
<div class="container-fluid">
    <div class="row">
        <!--  The sidebar -->
        <div class="sidebar col-sm-3    col-md-2    col-lg-2">
            <p>list of all CTS:</p>
            <ul class="nav nav-sidebar">
                <% if @all_cts %>
                <% @all_cts.each do |c| %>
                    <!-- Change default-tab here '? @action: "DEFAULT-ACTION"' -->
                    <li role="presentation" class=<%= "active" if @ctsname == c[:name] %> ><a href="<%= url("/forcts/#{c[:name]}/action/#{@action ? @action :
                        "db_config"}") %>"><%= c[:name] %></a></li>
                <% end %>
                <% end %>
            </ul>
        </div>

        <!-- CTS-Action-Bar -->
        <div class="main    col-sm-9 col-sm-offset-3    col-md-10 col-md-offset-2   col-lg-10 col-lg-offset-2 ">
            <div class="row">
                <div class="controlbar col-sm-12    col-md-12   col-lg-12">
                    <ul class="nav nav-tabs nav-controlbar"> <!-- nav-justified -->
                        <li role="presentation" class=<%= "active" if @action == "db_config" %>><a href="<%= url("/forcts/#{@ctsname}/action/db_config") %>">
                        DB-Config</a></li>
                        <li role="presentation" class=<%= "active" if @action == "import" %>><a href="<%= url("/forcts/#{@ctsname}/action/import") %>">Data
                        Import</a></li>
                        <li role="presentation" class=<%= "active" if @action == "servlet" %>><a href="<%= url("/forcts/#{@ctsname}/action/servlet") %>">
                        Servlet</a></li>
                        <li role="presentation" class=<%= "active" if @action == "browse" %>><a href="<%= url("/forcts/#{@ctsname}/action/browse") %>">Browse
                        Data</a></li>
                    </ul>
                </div>
            </div>

            <!-- Main Content -->
            <div class="row main_content">
                <%= erb @sub_erb if @sub_erb%>
            </div>
        </div>
```

Figure 6: step 1 in /views/main.erb

5.2 modification in cts_admin_root.rb

- now we have the add this action to the action handler
- this is done by adding another if-branch inside the action handler in our Sinatra class in cts_admin_root.rb
- for that find `get '/forcts/*/action/*' do |ctsname, action|`
- add a new branch for your action:

```
elsif @action == "myaction"
        # return the current cts name in upper case
        @my_data = @ctsname.upcase
```

```
@sub_erb = :myaction                    # show my site
```

- note1: @sub_erb specifies the erb-file (default path in /views) that should be rendered in the content area
- note2: the last statement "erb :main" will return a html-string (from /views/main.erb) which will in turn be rendered inside the layout.erb (default by name convention) at the yield statement
- note3: the layout.erb only holds all css and java-script declarations as well as some meta-tags. this file is used for every request. whereas the main.erb is the second-level layout. It is mostly used but not for the login-screen.
- note4: thus, the render structure is: layout.erb → main.erb → @sub_erb (here: myaction.erb)

```
446
447  # fixed url to select collections and actions
448  # matches /collection/hello/action/world
449  # also possible instead of block: params[:splat] # => [hello, world]
450  get '/forcts/*/action/*' do |ctsname, action|
451      # needs authentication
452      env['warden'].authenticate!
453      # get parameter
454      @ctsname    = ctsname
455      @action     = action
456
457      # config tab
458      if get_all_cts().map{ |e| e[:name] }.include?(ctsname)
459          if @action == "db_config"
460              # read the data base config part of the config file
461              @tbl_data   = read_config_between("", "#IMPORT#", @ctsname)
462              @sub_erb    = :cts_db_config    # show the config site
463          elsif @action == "import"
464              # read the import settings and the log-files of recent import activities. also check if an import is currently running
465              @tbl_data   = read_config_between("#IMPORT#", "#PARAMETERS#", @ctsname)
466              @import_runnig = File.exist?( @@import_rubylock )
467              @log_lines  = parse_ruby_logfile()
468              @sub_erb    = :cts_import       # show the import site
469          elsif @action == "servlet"
470              # read the servlet parameters
471              @tbl_data   = read_config_between("#PARAMETERS#", "", @ctsname)
472              @sub_erb    = :cts_servlet      # show the servlet site
473          elsif @action == "myaction"
474              # return the cts_name in upper case
475              @my_data    = @ctsname.upcase
476              @sub_erb    = :myaction     # show my site
477          elsif @action == "browse"
478              @capa       = "something went wrong"
479              begin
480                  tc      = tomcat_conf()
481                  cts_url = "http://" + tc[:host] + ":" + tc[:port] + "/" + @ctsname + "/cts/?request=GetCapabilities"
482                  site    = Nokogiri::HTML( open( cts_url ).read )
483                  @capa   = CGI.escapeHTML(site.to_xml)
484              rescue
485                  @capa = "CTS is not responding. #{cts_url}"
486              end
487              @sub_erb    = :cts_browse       # browse
488          else
489              @sub_erb    = :index
490          end
491      else
492          @sub_erb = :index
493      end
494
495      # show
496      erb :main
497  end
```

Figure 7: step 2 in cts_admin_root.rb

5.3 creation of /views/myaction.erb

- this is similar to the creation as for the menu bar (see section above)

5.4 optional: Javascripts

- this is similar to the creation as for the menu bar (see section above)