

Generating Documentation

Patricio

August 7, 2017

Contents

This file should be used to automatically generate the documentation of the `org-mode-clj-tests-utils` application. This file should reside in the `org` folder where all the org files are available.

What this file does is to define the Org-mode project for which we want to generate the documentation. The only thing you have to do is to run the code in each code block using `C-c` and then Org-mode will export each of the Org file into the appropriate export format.

Generate HTML Documentation

All the Org files that generate the `org-mode-clj-tests-utils` project can be exported as HTML documentation pages by running the following two code blocks.

But first let's take a look at how the Org-mode project that generate this documentation got configured:

- Settings
 - `:base-directory`
 - * The base directory is the current directory which is the `[project]/org/` directory where all the Org files are defined
 - `:recursive`
 - * We specify that we want Org-mode to generate HTML files for each Org file in all children folder (recursively)
 - `:publishing-directory`
 - * We specify where we want to publish the HTML documentation from the Org files

- `:publishing-function`
 - * We specify that we want to publish everything in HTML
- `:section-numbers`
 - * We don't want any kind of section numbers generated by Org-mode
- `:with-toc`
 - * We want to include a table of content for each generated documentation file
- `:auto-sitemap`
 - * We want to generate a sitemap automatically. The file is named `sitemap.html`
- `:html-head`
 - * We want to specify a style sheet that will be used by each generated HTML file. It should be located in `doc/html/css/`

Themes

Different themes and styles can be defined for the generated HTML pages. To enable a theme, you simply have to select the proper `:html-head` setting.

The main themes comes from the `org-html-themes` extension.

Publishing Options

Additional settings and configurations are available from these two web pages:

- Org-mode Publishing
- Org-mode 8 & 9 Updates

Publish

To publish in HTML, you simply have to run the following code blocks:

```
(defun org-publish-org-sitemap-includes (project &optional sitemap-filename)
  "Create a sitemap of pages in set defined by PROJECT.
  Optionally set the filename of the sitemap with SITEMAP-FILENAME.
  Default for SITEMAP-FILENAME is 'sitemap.org'."
  (let* ((project-plist (cdr project))
```

```

(dir (file-name-as-directory
      (plist-get project-plist :base-directory)))
(localdir (file-name-directory dir))
(exclude-regexp (plist-get project-plist :exclude))
(files (nreverse
        (org-publish-get-base-files project exclude-regexp)))
(sitemap-filename (concat dir (or sitemap-filename "sitemap.org")))
(sitemap-title (or (plist-get project-plist :sitemap-title)
                    (concat "Sitemap for project " (car project))))
(sitemap-sans-extension
 (plist-get project-plist :sitemap-sans-extension))
(visiting (find-buffer-visiting sitemap-filename))
file sitemap-buffer)
(with-current-buffer
(let ((org-inhibit-startup t))
  (setq sitemap-buffer
    (or visiting (find-file sitemap-filename))))
  (erase-buffer)
  (insert (concat "#+TITLE: " sitemap-title "\n\n")))
  (while (setq file (pop files))
    (let ((link (file-relative-name file dir))
          (oldlocal localdir))
      (when sitemap-sans-extension
        (setq link (file-name-sans-extension link)))
      ;; sitemap shouldn't list itself
      (unless (equal (file-truename sitemap-filename)
                     (file-truename file))
        (let ((entry
              (org-publish-format-file-entry
               org-publish-sitemap-file-entry-format file project-plist)))
          (insert (concat "* " entry "\n"
                          "#+INCLUDE: " link "\n")))))
      (save-buffer))
    (or visiting (kill-buffer sitemap-buffer))))

(setq org-publish-project-alist
      '(("org-mode-clj-tests-utils--doc-html"
        :base-directory "."
        :publishing-directory "../doc/html"

```

```

:publishing-function org-html-publish-to-html
:section-numbers nil
      :recursive t
:exclude "fulldoc\\\.org\\|project\\\.org\\|tangle\\-all\\\.org\\|setup\\\.org\\|publish"
:with-toc t
:auto-sitemap t
:sitemap-function org-publish-org-sitemap-includes

; ReadTheOrg Theme
:html-head "<link rel=\"stylesheet\" type=\"text/css\" href=\"themes/styles/readtheorg/
<link rel=\"stylesheet\" type=\"text/css\" href=\"themes/styles/readtheorg/css/readtheorg
<script src=\"https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js\"></script>
<script src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js\"></script>
<script type=\"text/javascript\" src=\"themes/styles/lib/js/jquery.stickytableheaders.js\"></script>
<script type=\"text/javascript\" src=\"themes/styles/readtheorg/js/readtheorg.js\"></script>

(setq org-publish-use-timestamps-flag nil)

(setq org-export-html-style-include-scripts nil
      org-export-html-style-include-default nil)

(org-publish-all)

```

Generate API Documentation

It is also important to generate great API documentation. We can easily do this by using the Codox Clojure application by running the following code block:

```

(use 'codox.main)

(generate-docs {:output-path "doc/api"})

```

Generate PDF Documentation

All the Org-mode files that defines this project can be exported in PDF by running the following code blocks.

```

(setq org-publish-project-alist
      '(("org-mode-clj-tests-utils--doc-pdf"

```

```
:base-directory "."
:publishing-directory "../doc/pdf"
:publishing-function org-latex-publish-to-pdf
      :recursive t
:section-numbers nil
:with-toc t
:auto-sitemap t)))

(org-publish-current-project)
```