Liaison

Aziz Ben Ali

July 10, 2023

Contents

This manual is a reference guide for Liaison 0.5.0, first released on 2023-01-06.

- Homepage: <`https://liaison.grtcdr.tn`>
- Repository: <`https://git.sr.ht/~grtcdr/liaison`>

1 Copying

2 Introduction

Liaison generates URLs that point to a specific page in a Git web interface, e.g. the tree, blob, blame, edit pages and more.

It aims to provide your readers with complete transparency about the history of changes (so long as you don't rewrite it) and the origin of web pages without hindering the reading experience.

It currently supports Sourcehut, Github, Gitlab, Gitea, cgit, including custom instances of the aforementioned services.

Documentation is available in a number of formats:

- HTML
- PDF
- Texinfo

3 Installation

Find and navigate to your desired project, and run the following in a shell prompt:

```
git clone https://github.com/grtcdr/liaison.git
```

Likewise, you may choose to add it as a submodule:

```
git submodule add https://github.com/grtcdr/liaison.git
```

It is assumed that you have a publishing script, named hereafter `publish.el`; in this file, you should define the project specification as well as any other configuration variables or special functions that affect the output of the published website.

It is best that you initiate the publishing process using a `Makefile`.

In your publishing script, add the package to the `load-path` and then require it like so:

```
(add-to-list 'load-path "liaison")
(require 'liaison)
```

If you're interested in a more elaborate but less tedious method for installing this package, you should read my article on Implementing per-project package management for ox-publish projects.

4 Integration

This section outlines the different steps needed to integrate this package into your website, it assumes that you have already created a publishing script.

In the previous section you were told to import `liaison`, but you'll also need to import `ox-publish` and any other libraries you depend on, in your publishing script.

```
(require 'ox-publish)
```

4.1 Using macros

This example registers a new `blob` macro which will receive the URL to the blob page of the currently visited document.

```
(setq org-export-global-macros
      '(("blob" . "(eval (liaison-get-resource-url 'blob))")))
```

We can now call the `blob` macro in an Org document and it will expand to the string we assigned to it.

```
#+TITLE: Example

* Heading

To view this document in its raw format, follow this link: {{{blob}}}.
```

4.2 In the preamble/postamble

In your publishing script, you will want to redefine the `org-html-format-spec` function such as to extend it with your custom format strings.

If you intend to use any other format strings, such as the preset `%a` or `%e`, don't forget to add them, otherwise they won't be expanded.

```
(defun org-html-format-spec (info)
  "Return format specification for preamble and postamble."
  `((?b . ,(liaison-get-resource-url 'blob))
    (?m . ,(liaison-get-resource-url 'blame))
    (?t . ,(liaison-get-resource-url 'tree))
    (?l . ,(liaison-get-resource-url 'log))
    (?p . ,(liaison-get-resource-url 'plain))
    (?e . ,(liaison-get-resource-url 'edit))))
```

This next step involves preparing your project structure to support the use of HTML templates; snippets which will be added onto our Org documents. These templates can reference at any point any of the format strings defined in the `org-html-format-spec` function.

The following is an example of such a template:

```
<nav>
  <a href="%e"><button>Edit</button></a>
  <a href="%m"><button>Blame</button></a>
  <a href="%b"><button>Blob</button></a>
  <a href="%t"><button>Tree</button></a>
  <a href="%l"><button>Log</button></a>
  <a href="%p"><button>Plain</button></a>
</nav>
```

During the export phase, Org will locate the format strings, and interpolate them using the return value of their associated function.

We'll now need a way to access the contents of these HTML files from our publishing script - to achieve that, you could use something like this:

```
(defun read-template (filename)
  "Read contents of FILENAME from the templates directory."
    (with-temp-buffer
      (insert-file-contents
       (file-name-concat "templates" filename))
      (buffer-string)))
```

We will use `read-template` along with the `:html-preamble` property to inject a preamble into the files of a particular project component.

```
(setq org-publish-project-alist
      (let ((preamble (read-template "preamble.html")))
(list
 (list "articles"
       :base-extension "org"
       :base-directory "articles"
       :publishing-directory "public/articles"
       :publishing-function 'org-html-publish-to-html
       :html-preamble preamble
       :html-postamble nil)
 (list "all"
       :components "articles"))))
```

Your website is now properly configured to use Liaison.

4.3 Publishing your website

The publishing script should be loaded before the `org-publish-project` function is called, this translates to the following command:

```
emacs --quick --batch \
      --load publish.el \
      --funcall org-publish-all t t
```

5 Custom instances

Liaison by default doesn't recognize domains beside the presets, so you'll have to specify the domain of your forge and its associated resource URL builder.

Now, suppose you're a member of freedesktop.org, your wonderful projects are present in the GitLab instance provided by your organization.

To add support for your infrastructure, you need to customize the `liaison-forge-alist` variable, here's an example:

```
(defvar liaison-forge-alist
  '(("gitlab.freedesktop.org" . #'liaison--build-gitlab-resource-url)))
```

6 Edge cases

6.1 Handling sitemaps

In some cases, you may find yourself using Liaison's functions in a publishing project with the `:auto-sitemap` option set to `t`.

Provided you are relying upon a templating system like the one proposed in this document, you will notice that when you visit the sitemap (e.g. `https://example.com/blog/sitemap.h` in a web browser, the document will contain the same preamble/postamble as the files which it is indexing.

To get around this, you should use set the `:sitemap-function` option. Here's an example in which we hide the postamble from appearing in the sitemap:

```
(defun my/blog-sitemap-function (title list)
  "Custom site map function for the blog project."
  (concat "#+OPTIONS: html-postamble:nil\n"
  (org-publish-sitemap-default title list)))
```

Here's another example which references a setupfile:

```
(defun my/blog-sitemap-function (title list)
  "Custom site map function for the blog project."
  (concat "#+SETUPFILE: setup.org\n"
  (org-publish-sitemap-default title list)))
```

By now you will have noticed that the strings we're concatenating to the default sitemap function, i.e. `org-publish-sitemap-default`, follow the same syntax as any Org document.

The result of this endeavor is a persistent configuration of the project's sitemap with our own custom options.