Liara Documentation

Matthäus G. Chajdas

Jun 06, 2020

Contents:

CHAPTER 1

Quickstart

If you want to get rolling with Liara right away, here's a quickstart guide to get you started.

1.1 Installation

Run:

```
pip install Liara
```

1.2 Create a project

Installing Liara deploys the command-line runner `liara`, which is your entry point for all future operations. To get started, create a new folder somewhere and run:

```
liara quickstart
```

This will deploy the required scaffolding for a super-simple blog, which we'll use throughout this guide.

1.3 Build the site

From the main directory, run `liara build` to build the site. The site will be generated in an `output` subdirectory by default.

1.4 Preview the site

Run `liara serve` to run a local web-server which will show the page. You can edit pages and templates while in this mode and hit refresh to see updates. You cannot add/remove content or change meta-data while running the

interactive server though.

1.5 Exploring the quick-start template

The quickstart produces a bunch of folders and files:

- `content` holds the content, which for the quickstart sample consists of a few blog posts. See *Content* for more details on how to structure the content.
- `templates` holds the template files. The quickstart sample uses jinja2 for its templates. `default.yaml` in that folder contains the default routes for the templates. See *Templates* for more information on how to use templates.
- `generators` contains a sample generator to create new blog posts. See *Generators* for more information.
- A few *configuration files*.

CHAPTER 2

Content

No site would be complete without content. In liara, content is provided in a separate directory, and mirrored into the output. That is, a document placed in `content-root/foo/bar.md` will be routed to `/foo/bar/index.html`. The content root can be set in the *Configuration*.

2.1 Documents

The bulk of the content are document nodes – Markdown or Html files which get processed by liara to Html and which get templates applied. liara supports some common Markdown extensions to handle tables and code snippets.

2.2 Metadata

Every document in liara is expected to start with a metadata header. A metadata header *must* contain at least the document title. Metadata can be provided as YAML or TOML. For YAML, use `---` as the delimiter, for TOML, use `+++`. Documents can be empty as long as the metadata is present, so this is a valid document with YAML metadata:

```
---
title: " This is an example"
---
```

You cannot mix the delimiters, i.e. using `---` to start and `+++` to end will result in a failure. Using more characters is also not supported.

2.3 Content filters

liara has some metadata which gets extra handling through content filters: `date` and `status`. `date` expects a timestamp, for example:

```
---
title: "My blog post"
date: 2096-11-22 19:30:56+01:00
---
```

Documents with dates beyond the current time the build is invoked will by default get filtered by the `DateFilter`. `status` can be used to hide content by setting it to `private` – which in turn will make the `StatusFilter` filter out the page. The filters can be set in the *Configuration*.

CHAPTER 3

Configuration

liara is driven through configuration files. The main file is `config.yaml`, which can reference other configuration files. To get the full default configuration, use `liara create-config`.

3.1 Directory settings

- `content_directory`: The root directory for all content. Output paths will be build relative to this folder.
- `resource_directory`: The folder containing resources, i.e. `SASS` or other files that need to get processed before they can be written to the output.
- `static_directory`: The folder containing static files, for instance downloads, images, videos etc.
- `output_directory`: The output directory.
- `generator_directory`: The folder containing *Generators*.

3.2 Build settings

- `build.clean_output`: If set to `True`, the output directory will be deleted on every build.
- `build.cache_directory`: The directory where the cache will be stored.

3.3 Content settings

- `content.filters`: Specifies which *content filters* will be applied while discovering content.
- `template`: The *template* definition to apply to the content.
- `collections`: Points to the file containing the *collection* definitions.
- `feeds`: Points to the file containing the *feed definitions*.

- `indices`: Points to the file containing the *index definitions*.
- `metadata`: Points to the file containing the *site metadata*.
- `relaxed_date_parsing`: If enabled, metadata fields named `date` will be processed twice. By default, liara assumes that `date` contains a markup-specific date field. If this option is on, and the `date` field is pointing at a string, liara will try to parse that string into a timestamp.
- `allow_relative_links`: Allow the usage of relative links in content files. This has a negative build time impact on any file containing relative links and is thus recommended to be left off.

3.4 Other settings

- `routes.static`: Points to the file containing *static routes*.

CHAPTER 4

Templates

Templates in liara are used to style content. There are two parts to every template, the first is the definition which describes which template should get applied to which URL, and the template execution environment which provides the content that will be consumed by the template.

4.1 Definition

Templates are defined using a template definition, which must contain at least two fields:

- `backend` chose the template engine, the default is `jinja2`.
- `paths` provides a dictionary containing key-value pairs. The key must be a *URL patterns*, the value the template file that should get applied for this pattern.

A very basic template could be defined as following:

```
backend: jinja2
paths:
  "/*": "default.jinja"
```

This would process any page using the `default.jinja` template.

Template definitions also support the following fields:

- `static_directory` specifies static files which will be deployed to the output. This can be used for images etc.
- `resource_directory` specifies resource files to be deployed, for instance `SASS` files.
- `image_thumbnail_sizes` is a dictionary which provides suffixes and the sizes to which images get resized. For instance, assume the following configuration:

  ```
  image_thumbnail_sizes:
    thumbnail: {width: 640}
  ```

This will resize any static image file (from the template or the site itself) to a maximum width of 640 pixels. The thumbnail will be stored using the same file path as the original, but with `thumbnail` added to the suffix. For instance, an input file named `foo.png` with width 800 would be resized to `foo.thumbnail.png` with a width of 640. Files which are below the size will get copied, so it's always safe to use the `.thumbnail` suffix.

Note: There is nothing special about `thumbnail` in the example above – any suffix can be used, and multiple suffixes are support.

4.2 Authoring Templates

Templates get applied to *`DocumentNode`* and *`IndexNode`* instances only. Inside a template, a few global variables are pre-populated to provide access to the site and page content. Note that the content of other nodes *cannot* be referenced inside a template (as the order in which they get executed is unspecified), however, metadata of other nodes *is* available and can be used.

- `page` references the current node, in form of a *`Page`* instance.
- `node` provides access to the current node directly, which will point to a *`Node`* instance.
- `site` provides access to the site in form of the *`SiteTemplateProxy`* object.

CHAPTER 5

Changelog

5.1 2.0

liara 2.0 is a complete rewrite of liara, with no shared code with the 1.x series. liara 2 is now template & content driven, and no longer just a library which simplifies static page generation. Unlike the 1.x series, it is possible to use liara 2 without writing any Python code.

CHAPTER 6

Collections

A collection in liara groups content together into an optionally ordered collection. This grouping is efficient, i.e. iterating a sorted collection does not incur any sorting cost. It is also efficient to look up the next/previous entry in an collection.

6.1 Definition

A collection definition consists of three elements:

- The name
- The filter – this is an *URL pattern* which defines all elements that are in this collection.
- The ordering – optionally specifies how the collection should be ordered. This is a metadata accessor, to access nested fields, separate the individual accesses using `.`. For instance, `date.year` will access the `date` metadata field first, and then the `year` attribute.

For example:

```
blog:
  filter: '/blog/**'
  order_by: 'date'
```

Defines a collection named `blog`, which contains all elements under `/blog`, ordered by the `date` metadata field.

6.2 Usage

Collections can be obtained from the *`Site`* object using *`get_collection()`*. If ordered, *`get_next_in_collection()`* and *`get_previous_in_collection()`* can be used to provide next/previous links.

The other use of collections are *Indices*.

CHAPTER 7

Indices

liara can generate indices for *Collections*, for instance, to create a post archive. This is a two step process – first, a collection needs to be defined, then an index can be built on top of that collection.

7.1 Definition

An index definition consists of:

- The name of the collection that is to be indexed.
- One or more grouping statements.
- The output path.

For example:

```
- collection: 'blog'
  group_by: ['date.year']
  path: '/blog/archive/%1'
```

This snippet defines a new index which groups based on the metadata field `date.year`, and produces paths of the form `/blog/archive/2017` etc. Multiple group-by fields can be used, in which case the collection will process them in turn. I.e. if grouped by year, month, first, all entries would get grouped by year, and then the entries in each separate year would get grouped by month. This requires two path components `%1` and `%2` to work, which will be consumed by each group in order.

A special syntax can be used for set-like fields, for instance tags. By adding a leading `*`, the group gets *splatted* into separate keys. For example, a page with a metadata field `tags` with the value `a, b`, grouped by `['*tags']` and using a path `/tags/%1` will be available under *both* `/tags/a` and `/tags/b`.

7.2 Usage

Indices can be targeted from *Templates*. An index node generated from an index will have the `references` attribute set, which allows iterating over all referenced nodes.

CHAPTER 8

Metadata

The default page metadata consists of:

- `title`: The site title.
- `description`: A description of the site.
- `base_url`: The default URL this page will get deployed to. liara may replace this with a different URL when serving the page locally.
- `language`: The language, provided as a language code of the form `en-US`.
- `copyright`: The default content copyright.

Metadata is used throughout liara, for instance, *Feeds* may use the metadata. It is available through `metadata` within liara, and via `metadata` to templates.

CHAPTER 9

Generators

Generators are used to create documents from the command line. Typically, they simplify the creation of metadata fields like `date`. To create a generator, you need to write some Python code. Place a new file into the generator directory (see *Configuration*), which has the same name as the document type you want to create. For instance, if you want to use `liara create blog-post`, you'd create a file named `blog-post.py`. This file must export a single function `generate(site: Site, configuration: Dict[Any, str]) -> pathlib.Path`. Invoking `liara create <type>` will first build the site, and then call the `generate` script to produce a new document. The function *must* return the location it stored the document to.

Warning: Generator scripts can execute arbitrary Python code. Be careful when using untrusted code.

CHAPTER 10

Feeds

Feeds provide a global overview over the site or a collection, typically in the form of a sitemap file or a RSS feed.

10.1 Definition

A feed definition consists of:

- The type of the feed to be generated.
- The output path.
- An optional limit on the number of elements that will be included in the feed.
- An optional collection to restrict the feed to.

For example:

```
rss:
  collection: blog
  path: /rss.xml
  limit: 10
json:
  collection: blog
  path: /feed.json
  limit: 10
sitemap:
  path: /sitemap.xml
```

This file will generate three feeds, one `RSSFeedNode` which uses the last 10 posts of the `blog` collection, one `JsonFeedNode` with the same configuration, and finally one `SitemapXmlFeedNode` for the whole site.

CHAPTER 11

Static routes

Static routes provide redirections so that content is reachable under multiple URLs. If static routes are configured, *RedirectionNode* instances are created for any source URL, and additionally, an Apache 2 `.htaccess` redirection file gets generated.

CHAPTER 12

URL patterns

In various places liara allows matching content by URL pattern. This is very similar to matching files using a glob pattern, the syntax is as following:

- `*` matches anything but a subpath (i.e., `/foo/*` will match `/foo/bar`, but not `/foo/bar/baz`)
- `**` matches anything recursively (i.e. `/foo/**` will match any path starting with `/foo/`)

Perfect matches take precedence over wildcard matches. That is, if there are two URL patterns `/foo/*` and `/foo/`, and they are matched against `/foo/`, both match but `/foo/` gets selected as it's a perfect match.

Additionally, URL patterns allow a query string to restrict the search to specific types. For instance, `/foo/**?kind=document` will match all *`DocumentNode`* below `/foo/`, but will ignore other node types. The nodes types that can be selected using this method are `document` for *`DocumentNode`* instances and `index` for *`IndexNode`* instances.

CHAPTER 13

liara

13.1 liara package

13.1.1 Submodules

13.1.2 liara.actions module

class liara.actions.**LinkType**

Bases: enum.Enum

An enumeration.

External = 2

Internal = 1

liara.actions.**gather_links**(*documents*, *link_type: liara.actions.LinkType*) → Dict[str, List[pathlib.PurePosixPath]]

Gather links across documents.

Returns A dictionary containing a link, and the list of document paths in which this link was found.

liara.actions.**validate_external_links**(*links: Dict[str, List[pathlib.PurePosixPath]]*)

Validate external links.

This issues a request for each link, and checks if it connects correctly. If not, an error is printed indicating the link and the documents referencing it.

liara.actions.**validate_internal_links**(*links: Dict[str, List[pathlib.PurePosixPath]], site: liara.site.Site*)

Validate internal links.

For each link, check if it exists inside the provided site. If not, an error is printed indicating the link and the documents referencing it.

13.1.3 liara.cache module

class `liara.cache.`**`Cache`**

Bases: `object`

A key-value cache.

`contains`(*key: bytes*) → bool

Check if an object is stored.

Parameters **`key`** – The key to check.

Returns `True` if such an object exists, else `False`.

`get`(*key: bytes*) → object

Get a stored object.

Parameters **`key`** – The object key.

Returns An object if one exists. Otherwise, the behavior is undefined. Use *`contains()`* to check if an object exists.

`put`(*key: bytes*, *value: object*) → bool

Put a value into the cache using the provided key.

Parameters

- **`key`** – The key under which `value` will be stored.
- **`value`** – A pickable Python object to be stored.

Returns `True` if the value was added to the cache, `False` if it was already cached.

class `liara.cache.`**`FilesystemCache`**(*path: pathlib.Path*)

Bases: *`liara.cache.Cache`*

A *`Cache`* implementation which uses the filesystem to cache data.

This cache tries to load a previously generated index. Use *`persist()`* to write the cache index to disk.

`contains`(*key: bytes*) → bool

Check if an object is stored.

Parameters **`key`** – The key to check.

Returns `True` if such an object exists, else `False`.

`get`(*key: bytes*) → object

Get a stored object.

Parameters **`key`** – The object key.

Returns An object if one exists. Otherwise, the behavior is undefined. Use *`contains()`* to check if an object exists.

`persist`()

Persists this cache to disk.

This function should be called after the cache has been populated. On the next run, the constructor will then pick up the index and return cached data.

`put`(*key: bytes*, *value: object*) → bool

Put a value into the cache using the provided key.

Parameters

- **`key`** – The key under which `value` will be stored.

- **value** – A pickable Python object to be stored.

Returns `True` if the value was added to the cache, `False` if it was already cached.

class `liara.cache.`**MemoryCache**

Bases: *liara.cache.Cache*

An in-memory *Cache* implementation.

This cache stores all objects in-memory.

contains(*key: bytes*) → bool

Check if an object is stored.

Parameters **key** – The key to check.

Returns `True` if such an object exists, else `False`.

get(*key: bytes*) → object

Get a stored object.

Parameters **key** – The object key.

Returns An object if one exists. Otherwise, the behavior is undefined. Use *contains()* to check if an object exists.

put(*key: bytes*, *value: object*) → bool

Put a value into the cache using the provided key.

Parameters

- **key** – The key under which `value` will be stored.
- **value** – A pickable Python object to be stored.

Returns `True` if the value was added to the cache, `False` if it was already cached.

13.1.4 liara.cmdline module

`liara.cmdline.`**build**(*options*)

Build a site.

`liara.cmdline.`**cli**()

Entry point for the command line.

`liara.cmdline.`**create**(*options*)

Create a document.

`liara.cmdline.`**create_config**(*options*)

Create a default configuration.

`liara.cmdline.`**find_by_tag**(*options*)

Find pages by tag.

This searches the metadata for a 'tags' field, which is assumed to be a list of tags.

`liara.cmdline.`**list_content**(*options*)

List all content.

If `options.format` is set to `tree`, this will print the content as a tree. If `options.format` is set to `list`, this will produce a flat list instead.

`liara.cmdline.`**`list_tags`**(*options*)
List all tags.

This uses a metadata field named 'tags' and returns the union of all tags, as well as the count how often each tag is used.

`liara.cmdline.`**`quickstart`**(*options*)
Create a quickstart project.

`liara.cmdline.`**`serve`**(*options*)
Run a local development server.

`liara.cmdline.`**`validate_links`**(*options*)
Validate links.

13.1.5 liara.config module

`liara.config.`**`create_default_configuration`**() → Dict[str, Any]
Creates a dictionary with the default configuration.

`liara.config.`**`create_default_metadata`**() → Dict[str, Any]
Creates a dictionary with the default metadata.

13.1.6 liara.feeds module

class `liara.feeds.`**`FeedNode`**(*path*)
Bases: `liara.nodes.GeneratedNode`

class `liara.feeds.`**`JsonFeedNode`**(*path*, *site: liara.site.Site*, *, *collection=''*, *limit=10*)
Bases: `liara.feeds.FeedNode`

A JSONFeed based feed.

`generate`()
Generate the content of this node.

After this function has finished, `self.content` must be populated with the generated content.

class `liara.feeds.`**`RSSFeedNode`**(*path*, *site: liara.site.Site*, *, *collection=''*, *limit=10*)
Bases: `liara.feeds.FeedNode`

A RSS 2.0 based feed.

`generate`()
Generate the content of this node.

After this function has finished, `self.content` must be populated with the generated content.

class `liara.feeds.`**`SitemapXmlFeedNode`**(*path*, *site: liara.site.Site*)
Bases: `liara.feeds.FeedNode`

A Sitemap 0.90 based feed.

`generate`()
Generate the content of this node.

After this function has finished, `self.content` must be populated with the generated content.

13.1.7 liara.md module

class liara.md.**HeadingLevelFixupExtension**(***kwargs*)

Bases: markdown.extensions.Extension

Markdown extension for the *HeadingLevelFixupProcessor*.

extendMarkdown(*md*)

Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

class liara.md.**HeadingLevelFixupProcessor**(*md=None*)

Bases: markdown.treeprocessors.Treeprocessor

This processor demotes headings by one level.

By default, Markdown starts headings with <h1>, but in general the title will be provided by a template. This processor replaces each heading with the next-lower heading, and adds a demoted class.

run(*root*)

Subclasses of Treeprocessor should implement a *run* method, which takes a root ElementTree. This method can return another ElementTree object, and the existing root ElementTree will be replaced, or it can modify the current tree and return None.

13.1.8 liara.nodes module

class liara.nodes.**DataNode**(*src*, *path*)

Bases: *liara.nodes.Node*

A data node.

Data nodes consist of a dictionary. This can be used to store arbitrary data as part of a *liara.site.Site*, and make it available to templates (for instance, a menu structure could go into a data node.)

class liara.nodes.**DocumentNode**(*src*, *path*, *metadata_path=None*)

Bases: *liara.nodes.Node*

load()

Load the content of this node.

publish(*publisher: liara.nodes.Publisher*) → pathlib.Path

Publish this node using the provided publisher.

reload()

Reload this node from disk.

By default, this just forwards to _load().

set_fixups(*, *load_fixups*, *process_fixups*) → None

Set the fixups that should be applied to this document node. The fixups should be set *before* calling *load()*.

Parameters

- **load_fixups** – These functions will be executed before *load()* returns.

- **process_fixups** – These functions will be executed before `process()` returns.

validate_metadata()

class liara.nodes.**DocumentNodeFactory**(*configuration*)
Bases: *liara.nodes.NodeFactory*

A factory for document nodes.

class liara.nodes.**FixupDateTimezone**
Bases: `object`

Set the timezone of the `metadata['date']` field to the local timezone if no timezone has been set.

class liara.nodes.**GeneratedNode**(*path*, *metadata={}*)
Bases: *liara.nodes.Node*

generate() → None
Generate the content of this node.

After this function has finished, `self.content` must be populated with the generated content.

publish(*publisher: liara.nodes.Publisher*)
Publish this node using the provided publisher.

class liara.nodes.**HtmlDocumentNode**(*src*, *path*, *metadata_path=None*)
Bases: *liara.nodes.DocumentNode*

A node representing a Html document.

process(*cache: liara.cache.Cache*)
Some nodes – resources, documents, etc. need to be processed. As this can be a resource-intense process (for instance, it may require generating images), processing can cache results and has to be called separately instead of being executed as part of some other operation.

By convention this method should populate `self.content`.

class liara.nodes.**IndexNode**(*path*, *metadata={}*)
Bases: *liara.nodes.Node*

An index node.

Index nodes are created for every folder if there is no `_index` node present, and from indices. An index node can optionally contain a list of references, in case the referenced nodes by this index are not direct children of this node.

add_reference(*node*)
Add a reference to an arbitrary node in the site.

publish(*publisher*) → pathlib.Path
Publish this node using the provided publisher.

references = None
Referenced nodes by this index.

An index can not rely on using `children` as those have to be below the path of the parent node. The `references` list allows to reference nodes elsewhere in the site.

class liara.nodes.**MarkdownDocumentNode**(*src*, *path*, *metadata_path=None*)
Bases: *liara.nodes.DocumentNode*

A node representing a Markdown document.

process(*cache: liara.cache.Cache*)

Some nodes – resources, documents, etc. need to be processed. As this can be a resource-intense process (for instance, it may require generating images), processing can cache results and has to be called separately instead of being executed as part of some other operation.

By convention this method should populate `self.content`.

class liara.nodes.**MetadataKind**

Bases: `enum.Enum`

An enumeration.

Toml = 3

Unknown = 1

Yaml = 2

class liara.nodes.**Node**

Bases: `object`

add_child(*child: liara.nodes.Node*) → None

Add a new child to this node.

The path of the child node must be a sub-path of the current node path, with exactly one more component. I.e. if the current node path is `/foo/bar`, a node with path `/foo/bar/baz` can be added as a child, but `/baz/` or `/foo/bar/boo/baz` would be invalid.

children

A list containing all direct children of this node.

get_child(*name*) → Optional[liara.nodes.Node]

Get a child of this node.

Returns The child node or `None` if no such child exists.

get_children(*, *recursive=False*) → Iterable[liara.nodes.Node]

Get all children of this node.

This function differs from *`select_children()`* in two important ways:

- It returns a list of *`Node`* instances and does not wrap it in a *`Query`*
- It can enumerate all children recursively.

kind = None

The node kind, must be set in the constructor.

metadata = None

Metadata associated with this node.

parent = None

The parent node, if any.

path = None

The output path, relative to the page root.

All paths *must* start with `/`.

process(*cache: liara.cache.Cache*) → None

Some nodes – resources, documents, etc. need to be processed. As this can be a resource-intense process (for instance, it may require generating images), processing can cache results and has to be called separately instead of being executed as part of some other operation.

By convention this method should populate `self.content`.

select_children()
Select all children of this node and return them as a *Query*.

src = None
The full path to the source file.

This is an OS specific path object.

class liara.nodes.**NodeFactory**
Bases: typing.Generic

A generic factory for nodes, which builds nodes based on the file type.

create_node(*suffix: str, src: pathlib.Path, path: pathlib.PurePosixPath, metadata_path: Optional[pathlib.Path] = None*) → T
Create a node using the provided parameters.

known_types

register_type(*suffixes: Union[str, Iterable[str]], node_type: type*) → None
Register a new node type.

Parameters

- **suffixes** – Either one suffix, or a list of suffixes to be registered for this type. For instance, a node representing an image could be registered to [.jpg, .png].
- **node_type** – The type of the node to be created.

class liara.nodes.**NodeKind**
Bases: enum.Enum

An enumeration.

Data = 4

Document = 3

Generated = 6

Index = 2

Resource = 1

Static = 5

class liara.nodes.**Publisher**
Bases: object

A publisher produces the final output files, applying templates etc. as needed.

publish_document(*document: liara.nodes.DocumentNode*)
Publish a document node.

publish_generated(*generated: liara.nodes.GeneratedNode*)
Publish a generated node.

publish_index(*index: liara.nodes.IndexNode*)
Publish an index node.

publish_resource(*resource: liara.nodes.ResourceNode*)
Publish a resource node.

publish_static(*static: liara.nodes.StaticNode*)
Publish a static node.

class liara.nodes.**RedirectionNode**(*path: pathlib.PurePosixPath*, *dst: pathlib.PurePosixPath*)
Bases: *liara.nodes.GeneratedNode*

A redirection node triggers a redirection to another page.

This node gets processed into a simple web site which tries to redirect using both `<meta http-equiv="refresh">` and Javascript code setting `window.location`.

generate()
Generate the content of this node.

After this function has finished, `self.content` must be populated with the generated content.

class liara.nodes.**ResourceNode**(*src*, *path*, *metadata_path=None*)
Bases: *liara.nodes.Node*

A resource node applies some process when creating the output.

This is useful if you have content where the source cannot be interpreted, and requires some process first before it becomes usable – for instance, `SASS` to `CSS` compilation.

publish(*publisher: liara.nodes.Publisher*) → pathlib.Path
Publish this node using the provided publisher.

reload() → None

class liara.nodes.**ResourceNodeFactory**
Bases: *liara.nodes.NodeFactory*

A factory for resource nodes.

class liara.nodes.**SassResourceNode**(*src*, *path*, *metadata_path=None*)
Bases: *liara.nodes.ResourceNode*

This resource node compiles `.sass` and `.scss` files to CSS when built.

process(*cache: liara.cache.Cache*)
Some nodes – resources, documents, etc. need to be processed. As this can be a resource-intense process (for instance, it may require generating images), processing can cache results and has to be called separately instead of being executed as part of some other operation.

By convention this method should populate `self.content`.

reload() → None

class liara.nodes.**StaticNode**(*src*, *path*, *metadata_path=None*)
Bases: *liara.nodes.Node*

A static data node.

Static nodes are suitable for large static data which never changes, for instance, binary files, videos, images etc.

is_image
Return `True` if this static file is pointing to an image.

publish(*publisher: liara.nodes.Publisher*) → pathlib.Path
Publish this node using the provided publisher.

update_metadata() → None
Update metadata by deriving some metadata from the source file, if possible.

For static nodes pointing to images, this will create a new metadata field `image_size` and populate it with the image resolution.

class liara.nodes.**ThumbnailNode**(*src*, *path*, *size*)
Bases: *liara.nodes.ResourceNode*

process(*cache: liara.cache.Cache*)

Some nodes – resources, documents, etc. need to be processed. As this can be a resource-intense process (for instance, it may require generating images), processing can cache results and has to be called separately instead of being executed as part of some other operation.

By convention this method should populate `self.content`.

`liara.nodes.`**extract_metadata_content**(*text: str*)

Extract metadata and content.

Metadata is stored at the beginning of the file, separated using a metadata seperation marker, for instance:

```
+++
this_is_toml = True
+++

content
```

This function splits the provided text into metadata and actual content.

`liara.nodes.`**fixup_date**(*document: liara.nodes.DocumentNode*)

If the date in the document is a string, try to parse it to produce a datetime object.

`liara.nodes.`**fixup_relative_links**(*document: liara.nodes.DocumentNode*)

Replace relative links in the document with links relative to the site root.

13.1.9 liara.publish module

class `liara.publish.`**DefaultPublisher**(*output_path: pathlib.Path*, *site: liara.site.Site*)

Bases: `liara.nodes.Publisher`

publish_generated(*generated: liara.nodes.GeneratedNode*)

Publish a generated node.

publish_resource(*resource: liara.nodes.ResourceNode*)

Publish a resource node.

publish_static(*static: liara.nodes.StaticNode*)

Publish a static node.

class `liara.publish.`**TemplatePublisher**(*output_path: pathlib.Path*, *site: liara.site.Site*, *template_repository: liara.template.TemplateRepository*)

Bases: `liara.publish.DefaultPublisher`

publish_document(*document*)

Publish a document node.

publish_index(*index: liara.nodes.IndexNode*)

Publish an index node.

13.1.10 liara.query module

class `liara.query.`**MetadataFilter**(*name*, *value=None*)

Bases: `liara.query.SelectionFilter`

Filter items which contain a specific metadata field and optionally check if that field matches the provided value.

match(*node: liara.nodes.Node*) → bool

Return `True` if the node should be kept, else `False`.

class liara.query.**MetadataSorter**(*item: str*, *reverse=False*, *case_sensitive=False*)

Bases: *liara.query.Sorter*

Sort nodes by metadata.

get_key(*item: liara.template.Page*)

Return the key to be used for sorting.

class liara.query.**Query**(*nodes*)

Bases: collections.abc.Iterable, typing.Generic

A query modifies a list of nodes, by sorting and filtering entries.

limit(*limit*) → liara.query.Query

Limit this query to return at most limit results.

reversed() → liara.query.Query

Return the results of this query in reversed order.

sorted_by_date(*, *reverse=False*) → liara.query.Query

Sort the entries in this query by the metadata field data.

sorted_by_metadata(*tag: str*, *, *reverse=False*, *case_sensitive=False*) → liara.query.Query

Sort the entries in this query by the specified metadata field.

sorted_by_title(*, *reverse=False*) → liara.query.Query

Sort the entries in this query by the metadata field title.

with_metadata(*name*, *value=None*) → liara.query.Query

Limit this query to only include nodes which contain the specific metadata field.

Parameters **value** – If value is provided, the field must exist and match the provided value.

with_tag(*name*) → liara.query.Query

Limit this query to only include nodes with a metadata field named tags which contains the specified tag name.

class liara.query.**SelectionFilter**

Bases: object

Base class for query selection filters.

match(*node: liara.nodes.Node*) → bool

Return True if the node should be kept, else False.

class liara.query.**Sorter**(*reverse=False*)

Bases: object

Base class for query sorters.

get_key(*item*)

Return the key to be used for sorting.

reverse

class liara.query.**TagFilter**(*name*)

Bases: *liara.query.SelectionFilter*

Filter items by a specific tag, this expects a metadata field named tags to be present, and that field must support checks for containment using in.

match(*node: liara.nodes.Node*) → bool

Return True if the node should be kept, else False.

13.1.11 liara.quickstart module

`liara.quickstart.`**`generate`**`()`

`liara.quickstart.`**`generate_configs`**`()`

`liara.quickstart.`**`generate_content`**`()`

`liara.quickstart.`**`generate_css`**`()`

`liara.quickstart.`**`generate_generator`**`()`

`liara.quickstart.`**`generate_templates`**`()`

`liara.quickstart.`**`generate_theme`**`()`

13.1.12 liara.server module

class `liara.server.`**`HttpServer`**(*site: liara.site.Site*, *template_repository: liara.template.TemplateRepository*, *configuration*, **, open_browser=True*)

Bases: `object`

`serve``()`

Serve the site with just-in-time processing.

This does not build the whole site up-front, but rather builds nodes on demand. Nodes requiring templates are rebuilt from scratch every time to ensure they're up-to-date. Adding/removing nodes while serving will break the server, as it will not get notified and re-discover content.

13.1.13 liara.site module

class `liara.site.`**`Collection`**(*site*, *pattern*, *order_by=[]*)

Bases: `object`

A collection is a set of nodes. Collections can be ordered, which allows for next/previous queries.

`get_next`(*node*)

Get the next node in this collection with regard to the specified order, or `None` if this is the last node.

`get_previous`(*node*)

Get the previous node in this collection with regard to the specified order, or `None` if this is the first node.

`nodes`

Get the (sorted) nodes in this collection.

class `liara.site.`**`ContentFilter`**

Bases: `object`

Content filters can filter out nodes based on various criteria.

`apply`(*node: liara.nodes.Node*) → bool

Return `True` if the node should be kept, and `False` otherwise.

class `liara.site.`**`ContentFilterFactory`**

Bases: `object`

`create_filter`(*name: str*) → liara.site.ContentFilter

class liara.site.**DateFilter**

Bases: *liara.site.ContentFilter*

Filter content based on the metadata field `date`.

If the date is in the future, the node will be filtered.

apply(*node: liara.nodes.Node*) → bool

Return `True` if the node should be kept, and `False` otherwise.

reason

class liara.site.**Index**(*site*, *collection: liara.site.Collection*, *path: str*, *group_by=[]*, *, *create_top_level_index=False*)

Bases: `object`

An index provides an index for a collection, by generating index nodes for the collection.

create_nodes(*site*)

Create the index nodes inside the specified site.

class liara.site.**Site**

Bases: `object`

This class manages to all site content.

add_data(*node: liara.nodes.DataNode*) → None

Add a data node to this site.

add_document(*node: liara.nodes.DocumentNode*) → None

Add a document to this site.

add_generated(*node: liara.nodes.GeneratedNode*) → None

Add a generated node to this site.

add_index(*node: liara.nodes.IndexNode*) → None

Add an index node to this site.

add_resource(*node: liara.nodes.ResourceNode*) → None

Add a resource to this site.

add_static(*node: liara.nodes.StaticNode*) → None

Add a static node to this site.

create_collections(*collections*)

Create collections.

create_indices(*indices*)

Create indices.

create_links()

This creates links between parents/children.

This is a separate step so it can be executed after merging nodes from multiple sources, for instance themes. It is safe to call this function multiple times to create new links; nodes which already have a parent are automatically skipped.

create_thumbnails(*thumbnail_definition*)

Create thumbnails.

Based on the thumbnail definition – which is assumed to be a dictionary containing the suffix and the desired size – this function iterates over all static nodes that contain images, and creates new thumbnail nodes as required.

data = None
The list of all data nodes in this site.

documents = None
The list of all document nodes in this site.

generated = None
The list of all generated nodes in this site.

get_collection(*collection: str*) → liara.site.Collection
Get a collection.

get_next_in_collection(*collection: str*, *node: liara.nodes.Node*) → Optional[liara.nodes.Node]
Get the next node in a collection.

get_node(*path: pathlib.PurePosixPath*) → Optional[liara.nodes.Node]
Get a node based on the URL, or `None` if no such node exists.

get_previous_in_collection(*collection: str*, *node: liara.nodes.Node*) → Optional[liara.nodes.Node]
Get the previous node in a collection.

indices = None
The list of all index nodes in this site.

metadata = None
Metadata describing this site.

nodes
The list of all nodes in this site.

register_content_filter(*content_filter: liara.site.ContentFilter*)
Register a new content filter.

resources = None
The list of all resources nodes in this site.

select(*query: str*) → Iterable[liara.nodes.Node]
Select nodes from this site.

The query string may contain `*` to list all direct children of a node, and `**` to recursively enumerate nodes. Partial matches using `*foo` are not supported.

set_metadata(*metadata: Dict[str, Any]*) → None
Set the metadata for this site.

This overrides any previously set metadata. Metadata is accessible via the *`metadata`* attribute.

static = None
The list of all static nodes in this site.

urls
The list of all registered URLs.

class liara.site.**StatusFilter**
Bases: *liara.site.ContentFilter*

Filter content based on the metadata field `status`.

If `status` is set to `private`, the node will be filtered. The comparison is case-insensitive.

apply(*node: liara.nodes.Node*) → bool
Return `True` if the node should be kept, and `False` otherwise.

reason

13.1.14 liara.template module

class liara.template.**Jinja2Template**(*template*)
Bases: *liara.template.Template*

render(***kwargs*) → str

class liara.template.**Jinja2TemplateRepository**(*paths: Dict[str, str], path: pathlib.Path*)
Bases: *liara.template.TemplateRepository*

Jinja2 based template repository.

This class has extra magic internally to allow it to be pickled/unpickled, which is necessary for multiprocessing.

find_template(*url*, *site: liara.site.Site*) → liara.template.Template

class liara.template.**MakoTemplate**(*template*)
Bases: *liara.template.Template*

render(***kwargs*) → str

class liara.template.**MakoTemplateRepository**(*paths: Dict[str, str], path: pathlib.Path*)
Bases: *liara.template.TemplateRepository*

find_template(*url*, *site: liara.site.Site*) → liara.template.Template

class liara.template.**Page**(*node*)
Bases: object

A wrapper around DocumentNode and IndexNode for use inside templates.

Templates only get applied to those node types, and the *Page* class provides convenience accessors while hiding the underlying node from template code.

content
Provides the content of this page.

meta
Provides the metadata associated with this page.

references
Provides the list of referenced nodes by this page.

This can be only used if the current page is an *IndexNode*, in all other cases this will fail. For index nodes, this will return the list of references as a *Query* instance.

url
Provides the current path of this page.

class liara.template.**SiteTemplateProxy**(*site: liara.site.Site*)
Bases: object

A wrapper around Site for use inside templates.

data
Get the union of all *liara.nodes.DataNode* instances in this site.

get_collection(*collection: str*) → query.Query
Get a collection in form of a *liara.query.Query* for further filtering/sorting.

get_next_in_collection(*collection: str*, *page: liara.template.Page*) → Optional[liara.template.Page]
Given a collection and a page, return the previous page in this collection or None if this is the first page.

get_previous_in_collection(*collection: str, page: liara.template.Page*) → Optional[liara.template.Page]

Given a collection and a page, return the next page in this collection or `None` if this is the last page.

metadata

Provide access to the metadata of this site.

select(*query*) → query.Query

Run a query on this site.

class liara.template.**Template**

Bases: `object`

render(***kwargs*)

class liara.template.**TemplateRepository**(*paths: Dict[str, str]*)

Bases: `object`

find_template(*url: str*) → liara.template.Template

update_paths(*paths: Dict[str, str]*)

13.1.15 liara.util module

liara.util.**add_suffix**(*path: pathlib.Path, suffix*)

Add a suffix to a path.

This differs from `with_suffix` by adding a suffix without changing the extension, i.e. adding `en` to `foo.baz` will produce `foo.en.baz`.

liara.util.**create_slug**(*s: str*) → str

liara.util.**flatten_dictionary**(*d, sep='.', parent_key=None*)

Flatten a nested dictionary. This uses the separator to combine keys together, so a dictionary access like `['a']['b']` with a separator `'.'` turns into `'a.b'`.

liara.util.**local_now**() → datetime.datetime

liara.util.**pairwise**(*iterable*)

For a list `s`, return pairs for consecutive entries. For example, a list `s0`, `s1`, etc. will produce `(s0,s1), (s1,s2), ...` and so on.

See: https://docs.python.org/3/library/itertools.html#recipes.

liara.util.**readtime**(*wordcount: int, words_per_minute=300*)

Given a number of words, estimate the time it would take to read them.

Returns The time in minutes if it's more than 1, otherwise 1.

13.1.16 liara.yaml module

liara.yaml.**dump_yaml**(*data, stream=None*)

Dump an object to Yaml.

This is a helper function which tries to use the fast `CDumper` implementation and falls back to the native Python version on failure.

liara.yaml.**load_yaml**(*s*)

Load a Yaml document.

This is a helper function which tries to use the fast `CLoader` implementation and falls back to the native Python version on failure.

13.1.17 Module contents

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

Y