quire

Quire

Getty

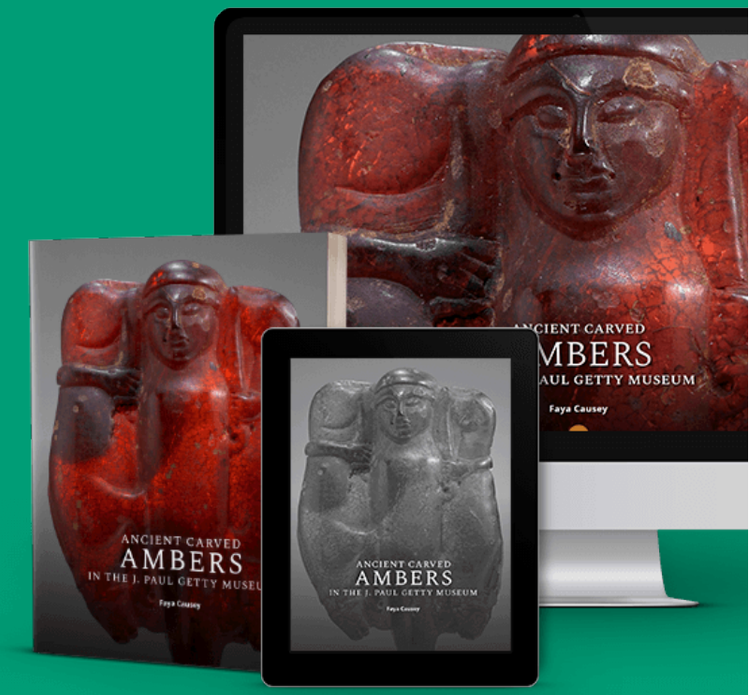
Quire is a modern, multiformat publishing tool designed to create books as authoritative and enduring as print and as vibrant and feature-rich as the web from a single set of plain text files. All without paying a fee or setting up and maintaining a complicated server.

Created by Getty, free to use.

https://quire.getty.edu

Quire

Create scholarly, visually rich digital books that last
Online · Print · E-book



Quire

Quire

J. PAUL GETTY TRUST, LOS ANGELES

What Is Quire?

Learn what Quire is, how it works, and whether it's right for you

Developed by Getty, Quire offers an elegant and affordable digital publishing solution ideal for creating dynamic publications in a variety of formats, including web, print, and e-book. In addition to being optimized for scholarly and visually rich publishing, Quire books are designed for longevity, sustainability, and discoverability.

Through the use of static site generation and plain text files, Quire makes it possible to preserve and distribute works in a stable and accessible format. Quire is flexible and feature-rich and can be used in its most basic configuration, or it can be customized, adapting to a vast array of use cases. Individuals at all levels of technical experience can use Quire. Our rapidly growing community includes scholars, curators, editors, designers, web developers, and the digitally curious.

Currently under development for potential release as open-source software, Quire builds upon Getty's expertise in traditional book publishing and is the cornerstone of the institution's pioneering digital publishing program. Through the development of Quire, Getty is carving a path forward for individuals and institutions looking to solve the challenges of creating digital books that are not only as beautiful as the print books we know and love but are equally as enduring.

Institutions currently publishing with Quire:

Ad Hoc Museum Collective / Duke University (Digital Art History & Visual Culture Research Lab) / / The Graphics Office / The University of Hong Kong (Libraries & Museum and Art Gallery) / J Paul Getty Trust (Museum & Publications) / Boston Public Library (Leventhal Map & Education Center) / Mills College Art Museum / Minneapolis Institute of Art / The Nelson-Atkins Museum of Art / Northwestern University (Libraries) / Emory University (Pitts Theology Library, Candler School of Theology) / San Jose Museum of Art / Tilt West

Quire has been used to produce:

Collection catalogues / exhibition catalogues / guides and toolkits / journals / online exhibitions / personal histories / proceedings volumes / reports / research projects / supplements to print publications

Features & Functionality

Multiformat Output
Generate publications as websites, EPUB and MOBI e-books, and PDFs

Realtime Publication Preview
Make changes and view them live in your browser

Web Accessibility
Include logical reading order, alt texts, language codes, semantic section elements, and more

SEO Optimization
Include metadata and tags for enhanced search engine optimization

Single-source Publishing
Change content once to change it across all formats, without duplicating effort

Customizable Styles & Fonts
Customize your publication's look and feel to suit your needs

Figure Images & Image Groups
Include high quality images in a variety of configurations

Zooming Images
Enable deep zooming capabilities with IIIF

Video & Audio Embeds
Insert audio and video clips (Soundcloud, YouTube, and Vimeo)

Dynamic Navigation
Use table of contents and menus to navigate and jump to other sections

Full-text Search
Perform a global search throughout your publication

Page-level Citation
Automatically generate citations to help readers reference your publication

Linkable Footnotes
Include footnotes that automatically link between note and text

Pop-up Bibliographies
Add pop-up citations and create bibliographies for a page or a whole book

Responsive Design
Web design adapts to smartphones, tablets, and desktops

Is Quire Right for You?

- Quire is developed, tested, and used by Getty. Getty is committed to Quire's continued use and growth in the field. Learn about Quire's history.
- Quire is free to use. No proprietary software or ongoing maintenance is required. Learn more about how quire works.
- E-book files are distribution-ready for Amazon, Apple, and other vendors; PDF files are print-on-demand ready, and the online edition can be hosted on any web server. Learn how to output and deploy your site.

- Quire is compatible with most operating systems, including macOS and Windows. See our installation guidelines for more information.
- No pre-existing technical skills are necessary. We are developing helpful tools and resources to support our users. Take our tutorial and browse our list of resources.
- We have a vibrant and diverse community of users who are available to answer your questions and provide guidance. Visit our Quire Community Forum.
- We are moving Quire towards an open-source, community-supported model, and are currently following open-source software standards. Learn why we choose an open-source model.
- For more detailed information about whether Quire is right for you, visit our Implementation Considerations.

Still not sure? Feel free to contact us for a one-on-one consultation to determine if Quire is right for you.

Documentation

Implementation Considerations

Is Quire right for you and your project?

Licensing

- For a free license to use Quire for your publication projects, and for additional access to support, please sign up.
- A Quire license enables users to create publications with Quire and distribute as they see fit. Users can modify the default theme in any way that suits their needs. However, we ask that users **don't redistribute Quire itself.**

Technology Requirements

- Quire is compatible with macOS and Windows. Currently, support is better for macOS.
- Installing Quire also requires installing a number of third-party applications. It is recommended to do so with care, and to consult with your institution's technology department if you're using a work machine.

- A standard web server / hosting service is required for publishing your project, but no special server software or setup is required.

Skill Requirements

- Quire does not offer a graphical user interface, but requires using a command-line shell and a text editor to create a publication. While this can be a new and somewhat intimidating way to work, we've found that most users can fairly quickly master the basics.
- Quire can be used "out of the box" by any individual, no matter their skill level, using Quire tutorials and documentation as support. However, in order to customize the layouts and features of the default Quire theme beyond what is provided out of the box, advanced knowledge of CSS, HTML, and static site generation is required.
- We do provide documentation and additional resources to aid non-technical users in gaining the skills and knowledge to use Quire in a more advanced way.

Affiliated Costs

Quire is free to use, however, there are occasionally associated costs:

- In order to output your Quire publication as a PDF you will need to use Prince XML, which requires purchasing a license for commercial or non-watermarked use. Learn more about Prince's pricing plan..
- If you are at a small institution, you may want to hire a developer for general technical support, design customization, and deployment.

Content Compatibility

- Quire is optimized for book-like projects. Especially those that rely on visual illustrations and/or scholarly apparatus like citations, bibliographies, and footnoting.
-

For those looking to publish more non-textual, image-intensive presentations, like an online exhibition or a fully illustrated children's book or graphic novel, Quire may not be the right fit.

- Using Quire for journal publishing is also an option, though not yet fully developed.

Versioning

- Quire is in limited beta. While fully functioning, changes and improvements are continuing to be made to it which may lead to incompatibility across versions.
- Quire projects, once published, no longer require Quire to keep running. After publication, a compatible version of Quire would only be necessary to make new updates and re-output the various file formats.

For Developers

API docs and Quire repository guide

Quire is in a limited beta and not yet released as open-source software. For a free license to use Quire for your publication projects, and for additional access to support, please sign up

Quire is centered around the static site generator, Hugo. Quire's command-line interface is written in JavaScript and requires Node.js LTS to run.

Repositories

quire

https://github.com/thegetty/quire

This repo contains the command-line interface for Quire, default starter content used as placeholder content when starting a new Quire project, and a default theme designed to broadly cover a full range of use-cases and demonstrate the range of Quire's content model.

quire-docs

https://github.com/thegetty/quire-docs

This repo is specifically for the Quire website and documentation.

Configuration

General configuration options for a project. See *Configure Hugo* for more options and information.

Location: `config.yml`, or any of the environment and format specific config files found in the `config` directory

Type: Object

Object Properties	Expected Value	Description
`baseURL`	url	The base url for your project.
`blackfriday`	object	Options for Blackfriday, Hugo's markdown renderer. Quire v0.18.0 and below. See below.
`buildDrafts`		
`canonifyURLs`	boolean	Converts all internal links to being in complete canonical format. Default is `false`.
`disableKinds`		
`footnoteReturnLinkContents`	string	Controls the appearance of the link added to the end of footnotes. Default is "↩".
`googleAnalytics`		
`languageCode`		
`markup`	object	Options for Hugo's markdown rendering. Quire v0.19.0 and above. See below.

Object Properties	Expected Value	Description
`metaDataFormat`	“yaml”, “toml”, “json”	Default is “yaml”.
`params`	object	Additional parameters for Quire. See below.
`pluralizeListTitles`		
`publishDir`		
`relativeURLs`	boolean	Keeps all internal links relative. Default is `true`.
`title`		
`theme`	url/id	The name of the theme, in the `theme` directory you’re using. Quire starter kit default is `quire-base-theme`.

`blackfriday`

Configuration for the Blackfriday markdown rendering engine.

Location: `config.yml`, or any of the environment and format specific config files found in the `config` directory

Type: Object

Compatibility: Quire v0.18.0 and below

Object Properties	Expected Value	Description
`fractions`	boolean	When set to `true` any numbers separated by a slash are automatically converted to fractions. Default is `false`. Though even then 1/4, 1/2 and 3/4 are still converted. Recommend always using proper unicode fractions: ¼, ½, ¾, ⅛, ⅜, ⅝, ⅞.
`hrefTargetBlank`	boolean	When set to true, any Markdown links to external pages and resources will open in a new tab/window for the user.

`markup`

Configuration for the markdown rendering engine. See Hugo's Markup Configuration for more options and information.

Location: `config.yml`, or any of the environment and format specific config files found in the `config` directory

Type: Object

Compatibility: Quire v0.19.0 and above

Object Properties	Expected Value	Description
`goldmark.renderer.unsafe`	boolean	"true" (default) or "false", allows the inclusion of HTML markup in markdown pages.

`params`

Quire-specific project parameters.

Location: `config.yml`, or any of the environment and format specific config files found in the `config` directory

Type: Object

Parameter	Expected Value	Description
`searchEnabled`	boolean	Turn on or off the built-in text search capability for users.
`licenseIcons`	boolean	Whether or not to display Creative Commons license icons.
`pageLabelDivider`	string	". " default, determines the text/ spacing to be inserted between page .label and page .title.
`citationPageLocationDivider`	string	", " default, determines the text/ spacing to be inserted between the citation and the page number in the q-cite shortcode.
`displayBiblioShort`	boolean	Whether a bibliography generated with the q-cite or q-bibliography shortcodes should display the short form of the reference, along with the long.
`biblioHeading`		
`imageDir`	string	"img" default, the directory in the `/static/` directory where you put your images.
`tocType`	"full", "short"	"short" will hide all sub-section pages.
`menuType`	"full", "short"	"short" will hide all sub-section pages.
`prevPageButtonText`	string	"Back" default.

Parameter	Expected Value	Description
`nextPageButtonText`	string	“Next” default.
`entryPageSideBySideLayout`	boolean	Entry pages can have a side-by-side layout with image on the left and text on the right, this can be controlled by `class: side-by-side` in the page YAML, or globally with this parameter.
`entryPageObjectLinkText`	string	“View in Collection” default.
`figureLabelLocation`	“on-top”, “below”	Whether the figure label is “on-top” of the image in the upper left corner, or “below” it with the caption.
`figureModal`	boolean	If figures should be clickable to open into a full-screen modal window.
`figureModalIcons`	boolean	Whether to display icons with the figure modal links.
`figureZoom`	boolean	Whether figures should zoom or not inside the modal.
`citationPopupStyle`	“icon”, “text”	Displays the citation pop-up links with an icon, or without.

Publication API

Location: `publication.yml`

Type: Object

Object Properties	Expected Value	Description
`title`	string	The title of your publication.
`subtitle`	string	The subtitle of your publication.
`short_title` *	string	A short version of your title, primarily for use in navigation elements with limited space.
`reading_line`	string	An additional title line for your publication.
`url` *	url	The full URL of your final publication.
`pub_type`	“book”, “journal-periodical”, “other”	Can be one of three values. Determines how key search-engine metadata is defined.
`pub_date`	YYYY-MM-DD	The first date your publication will be released.
`language`	2-letter ISO 639-1 language code(s)	Taken from the the list at https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes. List multiple languages using a comma-separated list.
`identifier`	object	See below.
`publisher`	array	See below.
`contributor`	array	See below.
`contributor_as_it_appears`	string	
`promo_image`	url	
`description`	object	See below.

Object Properties	Expected Value	Description
`subject`	array	See below.
`copyright`	string	
`license`	object	See below.
`resource_link`	array	See below.
`revision_history`	array	See below.
`repository_url`	url	A public repository of the source code and revision history for the publication.
`series_periodical_name` *	string	
`series_issue_number`	string	

`publisher`

Location: `publication.yml`

Type: Array

Item Attributes	Expected Value	Description
`name`	string	Name of the publisher.
`location`	string	Publisher location, city.
`url`	url	Publisher homepage.
`logo`		

`description`

Location: `publication.yml`

Type: Object

Object Properties	Expected Value	Description
`one_line`	string	
`full`	string	
`online` *	string	The `online` and `pdf_ebook` fields allow you to add additional text to the `full` description that is specific to either the online, or the PDF/EPUB/MOBI editions and will only show up there. For instance, in order to point to special features in one or the other of the formats.
`pdf_ebook` *	string	

`subject`

Location: `publication.yml`

Type: Array

Item Attributes	Expected Value	Description
`type`	“keyword”, “bisac”, “getty”	
`name`	string	
`identifier`	string	

`license`

Location: `publication.yml`

Type: Object

Object Properties	Expected Value	Description
`name`	string	Name of the license.
`abbreviation`		If using a Creative Commons licenses, should match one of the seven available options: “CC0”, “CC BY”, “CC BY-SA”, “CC BY-ND”, “CC BY-NC”, “CC BY-NC-SA”, or “CC BY-NC-ND”.
`url`	url	Link to the license text.
`scope`	“text-only”, “full”, “some-exceptions”	
`icon` *	url	
`online_text`	string	Markdown okay. Will override the automatically generated license text for the online edition only.
`pdf_ebook_text`	string	Markdown okay. Will override the automatically generated license text for the PDF and e-book editions only.

`resource_link`

Location: `publication.yml`

Type: Array

Item Attributes	Expected Value	Description
`name`	string	How the link will be named.
`type`	“other-format”, “related-	

Item Attributes	Expected Value	Description
	resource", "footer-link"	
`media_type`	string	Taken from the list at https://www.iana.org/assignments/media-types/media-types.xhtml.
`link_relation`	string	Taken from the list at https://www.iana.org/assignments/link-relations/link-relations.xhtml.
`url`	url	URL to web resource or to download.
`identifier` *	object	See below.
`file_size_mb` *	integer	For downloads, file size in megabytes. Often appended to `name` in the interface, depending on your theme.
`icon` *	url	

`revision_history`

Location: `publication.yml`

Type: Array

Item Attributes	Expected Value	Description
`date`	YYYY-MM-DD	
`summary` *	list	

`identifier`

Location: `publication.yml` , in any `.Page.Params.` , or in any `resource_link`

Type: Object

Item Attributes	Expected Value	Description
`isbn`	10- or 13-digit ISBN	For use with `pub-type` of "book". ISBNs can be purchased individually or in packages at http://www.isbn.org/.
`issn`	8-digit ISSN	For use with `pub-type` of "journal-periodical". ISSNs can be requested through https://www.issn.org/.
`doi` *	string	Not yet implemented.
`uuid` *	string	Not yet implemented.
`url`		Possibly replacing `url` in general Publication level??

`contributor`

Location: `publication.yml` or in any `.Page.Params.`

Type: Array

Item Attributes	Expected Value	Description
`id`	string	Numbers and lowercase letters only, with no spaces or special characters ("001", "fig-01a", "bird-picture", etc).
`type`	"primary", "secondary", or user choice	
`first_name`		All contributors must have either a first and last name, or a full name defined.
`last_name`		

Item Attributes	Expected Value	Description
`title`		
`affiliation`		
`url`	url	
`bio`		Markdown okay.
`pic`	url	Should be the file name of a JPG, PNG or GIF image (`fig01.jpg`). Avoid using spaces or special characters, and if it's in a sub-folder within the main `img` directory (which is defined by the `imageDir` parameter in the `config.yml` file), it should include that sub-folder name as well (`contributors/fig01.jpg`).
`full_name`		
`file_as`		

`figure_list`

Location: `figures.yml`

Type: Array

Item Attributes	Expected Value	Description
`id`	string	Numbers and lowercase letters only, with no spaces or special characters ("001", "fig-01a", "bird-picture", etc).
`src`	url	Should be the file name of a JPG, PNG or GIF image (`fig01.jpg`). Avoid using spaces or special characters, and if it's in a sub-folder

Item Attributes	Expected Value	Description
		within the main `img` directory (which is defined by the `imageDir` parameter in the `config.yml` file), it should include that sub-folder name as well (`comparatives/fig01.jpg`).
`alt`	string	For accessibility, all images should have alternative text descriptions. (Tips on crafting good alt text.) Only ever leave blank if the image is purely decorative.
`caption`	string	The caption to appear below the figure. Special characters are allowed. Use Markdown for formatting.
`credit`	string	Follows the caption. Markdown allowed.
`media_type`	“youtube”, “vimeo”	Currently supports video hosted on YouTube or Vimeo. (May eventually expand to HTML5 video, audio, and Soundcloud, and others.) When a `media_type` is defined, a `media_id` must be as well. For video, it is also recommended that an image `src` still be used (presumably being a screenshot from the video) so as to provide a fallback for PDF and EPUB output.
`media_id`	string	The ID of the video resource on YouTube or Vimeo. For example, in the URL https://www.youtube.com/watch?v=VYqDpNmnu8I, the `media_id` would be `VYqDpNmnu8I` ; and in https://vimeo.com/221426899 it is `221426899` .
`aspect_ratio`	“standard”, “widescreen”	For use with video `media_type` s to properly scale video embeds. When no value is provided, the default is “widescreen”.
`label`	string	

Item Attributes	Expected Value	Description
`download`	boolean	If "true", download icon will be added to image viewer, allowing users to easily download the image file. Currently only implemented in the page `type: entry` image viewer. Default is "false".

`entries`

Location: `references.yml`

Type: Array

Item Attributes	Expected Value	Description
`id`	string	
`short`	string	The short form of the citation, ie., Brown 1984. Markdown okay.
`full`	string	The full form of the citation. Markdown okay.
`sort`	string	

`object_list`

Location: `objects.yml`

Type: Array

Attribute	Expected Value	Description
`id`	string	Required. Used to reference objects from entry pages. Should be numbers and lowercase letters only, with no spaces or special characters (`001` , `fig-01a` , etc).
`figure`	array	A list of one or more images of the object. It is recommended that this list be only of `id` values corresponding with `id` s in your project's `figures.yml` file.
`link`	url	A URL link to a page with more/current information on the object. Usually the object in the museum's online collection pages.
`date_start` , `date_end`	integer	Reserved for future use in Quire.
`dimension_width` , `dimension_height` , `dimension_depth`	integer	Reserved for future use in Quire.

Objects also support arbitrary attributes, which might include `title` , `artist` , `collection` , etc. Those added will be output in a table on collection catalogue entry pages. The ordering of the display can be controlled with `object_display_order` in `objects.yml` . See *Guide on Collection Catalogues*.

Page API

Location: Any Markdown page in the `/content/` directory.

Type: Object

Attribute	Expected Value	Description
`label`	string	A label for the page “Chapter 1”, “2”, “III”, etc.
`title`	string	
`subtitle`	string	
`short_title`	string	Used in navigation items where a long title would be too unwieldy.
`type`	“page” (default), “essay”, “entry”, “cover”, “contents”, “splash”, or “data”	See *Define Page Types* for examples
`class`	string	Can accept any string, which will be included as a class in the main page element to facilitate style customization. A number of pre-defined classes also exist in the Quire Default Theme. Pages with `type: contents` can have class `list` (default), `brief`, `abstract`, or `grid`. Pages with `type: entry` can have class `landscape` (default) or `side-by-side`.
`weight`	integer	Controls ordering of pages in the publication.
`object`	array	See *Catalogue Entries*.
`contributor`	array	See *Contributors*.
`contributor_byline`	boolean	

Attribute	Expected Value	Description
`abstract`	string	Markdown okay.
`slug`	url path	Will change the URL of the page. Or use a period `.` to make the URL be the directory name (homepage). Read more in the *Page Types & Structure* chapter of this guide.
`toc`	boolean	Default is "true". Page will not display in contents page if "false".
`menu`	boolean	Default is "true". Page will not display in menu if "false".
`online`	boolean	Default is "true". Page will not display in the online edition if "false".
`pdf`	boolean	Default is "true". Page will not display in the PDF edition if "false".
`epub`	boolean	Default is "true". Page will not display in either the EPUB or MOPBI e-book editions if "false".
`image`	url	

Pages with `type: contents` can have class `list` (default), `brief`, `abstract`, or `grid`. Pages with `type: entry` can have class `landscape` (default) or `side-by-side`.

Shortcodes API

`q-class`

Used for styling. Wrapping any Markdown text in this shortcode will wrap it in a `<div>` with the given class name in the HTML output.

```
{{< q-class "" >}} {{< /q-class >}}
```

Positional Parameter[†]	Expected Value	Description
`0` (class name)	string	String without spaces or special characters to be used as a class for CSS styling.

q-bibliography

Generates a bibliography from the entries in the project's `bibliography.yml` file. See *Citations & Bibliographies*.

```
{{< q-bibliography sort="" >}}
```

Parameter	Expected Value	Description
`sort`	string	Optional. Value can be any string that matches a key in the entires under `entries` of the `references.yml` file, to indicate which key to alphabetically sort the output bibliography by. Without `sort` the default sort is on "full".

q-cite

Adds a linked Author Date citation reference to the text, and a hover pop-up with the full citation text. It also adds the citation to a map of cited works, which can then be output as a page-level bibliography on essay and entry type pages. See *Citations & Bibliographies*.

```
{{< q-cite "" "" "" >}}
```

Positional Parameter[†]	Expected Value	Description
`0` (author date reference)	string	Should exactly match an `id` value under `entries` in the `references.yml` file. Typically something like "Jones 1974".
`1` (page number)	string	Optional. A page number of the specific citation. Will be appended to the citation text with a text divider defined by `citationPageLocationDivider` in `config.yml`
`2` (display text)	string	Optional. Alternate text that should be displayed instead of the default Author Date provided in the first parameter.

`q-contributor`

Can be used to create a page of contributor biographies, a section of bios for a single page, a simple list of contributors, a byline for a particular page, or other similar outputs. See *Contributors*.

```
{{< q-contributor range="" format="" align="" >}}
```

Parameter	Expected Value	Description
range	"page", "all", or string	Defines which contributors to include. An arbitrary value matching a contributor type such as "primary" or "secondary" may also be used.
format	"initials", "name", "name-title", "name-title-block", "bio"	Defines what format the list should take.
align	"left" (default), "center", "right"	Optional. Defines how the output text will be aligned.

q-figure

Inserts a formatted figure image, label, caption and credit line. If using a `data/figures.yml` file, only an `id` parameter is required for this shortcode. If other values supplied directly in the shortcode they will override any corresponding values in the `data/figures.yml`. See *Figure Images* and `figure_list` above.

```
{{< q-figure id="" src="" label="" caption="" credit="" alt="" class="" >}}
```

Parameter	Expected Value	Description
`id`	string	Spaces or special characters should not be used and will be stripped out. When used in a shortcode *without* a corresponding `src` parameter, the shortcode will look for a matching `id` in the project's `data/figures.yml` file. When used in a shortcode *with* a corresponding `src` parameter, this will create an `id` for the image markup that can be used to link to the image directly (`mypublication.com/chapter01/#fig-3`) and ignores any potentially corresponding information in the `data/figures.yml` file.
`src`	url	Should be the file name of a JPG, PNG or GIF image (`fig01.jpg`). Avoid using spaces or special characters, and if it's in a sub-folder within the main `img` directory (which is defined by the `imageDir` parameter in the `config.yml` file), it should include that sub-folder name as well (`comparatives/fig01.jpg`). If your project uses `data/figures.yml` file, you shouldn't use a `src` parameter in the shortcode as it will override all other information.

Parameter	Expected Value	Description
`alt`	string	For accessibility, all images should have alternative text descriptions. (Tips on crafting good alt text.) Only ever leave blank if the image is purely decorative.
`caption`	string	The caption to appear below the figure. Special characters are allowed. Use Markdown for formatting.
`credit`	string	Follows the caption. Markdown allowed.
`label`	boolean	Default is set to `true`. `true` will add a label to the caption, such as "Figure 1.3", `false` will remove the label. The global label setting is in the `config.yml` file under the parameter `figureLabels`.
`class`	`is-pulled-right`, `is-pulled-left`, `is-full-width`, `is-centered-small`	Sets the style of the figure image.

`q-figure-group`

Like `q-figure`, but with handling for multiple images at once. See *Figure Images* and `figure_list` above.

```
{{< q-figure-group id=" , , " grid="" src="" label="" caption="" credit="" class="" >}}
```

Parameter	Expected Value	Description
`id`	string	One or more comma-separated `id`s that match corresponding values in the project's `data/figures.yml` file.
`caption`	string	The caption to appear below the figure group. Special characters are allowed. Use Markdown for formatting. Overrides any caption information provided in `data/figures.yml`.
`credit`	string	Follows the caption. Markdown allowed. Overrides any caption information provided in `data/figures.yml`.
`label`	boolean	Default is set to `true`. `true` will add a label to the caption, such as "Figure 1.3", `false` will remove the label. The global label setting is in the `config.yml` file under the parameter `figureLabels`. If a `caption` is also provided in the shortcode, the labels will be applied on their own directly under each image in the group, rather than as part of the caption.
`class`	`is-pulled-right`, `is-pulled-left`, `is-full-width`, `is-centered-small`	Sets the style of the group of figures overall.
`grid`	`1`, `2`, `3`, `4`, `5`, `6`	Determines the horizontal width (in number of images) of the image grid. If no grid is set, the images will stack on top of one another vertically.

`q-figure-zoom`

Uses IIIF to add high resolution images with deep-zoom capabilities. See *Zooming Images with IIIF*

Parameter	Expected Value	Description
id	string	

NOTES

* Attributes with an asterisk are in the process of being reviewed as they may not be currently being used and/or may be deprecated.

† Positional parameters are included in shortcodes without a name defining them. See `q-class`, and `q-cite`.

Install or Update

Get set up to use Quire on macOS, Window, or Linux

WARNING!

Quire is in a limited beta, © J. Paul Getty Trust, and not yet released as open-source software. For a free license to use Quire for your publication projects (and for additional access to support) please sign up.

macOS Installation

Installing and running Quire requires using the Terminal command-line shell. Open it from your Applications/Utilities folder or by pressing Command–Space Bar and typing "Terminal". If you are new to the command-line, read our tutorial *Working in a Command-line Shell*.

Quick Install

If you're eager to get started, this will install the complete Quire package but without e-book or PDF output capability. These may be added later, by following steps 3–5 in the Full Install guidelines below.

1. In Terminal, install Apple's Xcode with: `xcode-select --install`

2. Download and install the **LTS** version of Node.js: https://nodejs.org
3. In Terminal, install Quire with: `npm install --global @thegetty/quire-cli`
4. Confirm installation by pulling up a list of Quire commands: `quire --help`

The Quire installation process may take a minute or two, during which time there will be messaging output in Terminal. The only messages of any concern are those labeled as ERROR or ERR. If you see these errors, or if you see "command not found" after entering `quire --help` in step 4., search our Discussions Forum to troubleshoot installation issues.

> **WARNING!**
>
> **The Quire installation process may take a minute or two, during which time there will be messaging output in Terminal. The only messages of any concern are those labeled as ERROR or ERR. If you receive an error message or get a "command not found" message after entering `quire --help` in step 4., visit our Discussions Forum to troubleshoot installation issues.**

Full Install

Follow the steps below to first install the support software for Quire, and then Quire itself.

1. **Apple's Xcode** is a set of developer tools for your mac. Install Apple's Xcode by copying and pasting the following command and pressing enter. If Xcode is not already installed, an additional notification will pop up. Click "Install" and follow the prompts.

```
xcode-select --install
```

2. **Node.js** enables you to run javascript on your computer. Visit the Node.js site, and download and install the older LTS (long-term support) version, which is sufficient and more stable than the higher "Current" version that is also available for download: https://nodejs.org.

3. **Pandoc** serves two purposes in Quire: You can use it to convert Word documents to Markdown, and it enables you to create EPUB e-book files of your Quire project with the `quire epub` command. Download the macOS installer, double-click it and follow the prompts to install: https://pandoc.org/installing.html

4. **Kindle Previewer**, along with Pandoc, enables you to create MOBI e-book files of your Quire project with the `quire mobi` command. Visit Amazon's Kindle Previewer page and download the Mac version: https://kdp.amazon.com/en_US/help/topic/G202131170. Install by double clicking the icon and following the prompts.

5. **PrinceXML** enables you to create a PDF version of your Quire project with the `quire pdf` command. Visit the PrinceXML site, download the Mac OS version, uncompress the folder, and rename it to "prince": http://www.princexml.com/download/.

   Open Terminal and type these lines in, hitting enter after each:

   ```
   cd Downloads/prince
   sudo ./install.sh
   ```

   You will be prompted to enter your computer password. Press enter. You should receive a message that PrinceXML will be installed in the `/usr/local` directory. Press enter again. If successful you will see a message in the Terminal saying "installation complete." Or, if you get a "no such file or directory" message after the first line, the file you dowloaded is either not in your Downloads folder, or is not named "prince". Correct as necessary and try again.

When complete, type `cd` into the Terminal to return to your home directory.

```
cd
```

6. **Quire** is operated through a command-line interface (CLI) that enables you to create, preview and output publications using Terminal with commands like `quire new`, `quire preview` and `quire site`. You can learn more in the *Quire Commands* chapter of our guide. Copy and paste the following line into Terminal to install Quire:

   ```
   npm install --global @thegetty/quire-cli
   ```

   The Quire installation process may take a minute or two, during which time there will be messaging output in Terminal. The only messages of any concern are those labeled as ERROR or ERR. These likely indicate a failed installation.

   To verify installation, enter the command below. This will give you a list of commands that will help you get started using and navigating Quire. And if you get "command not found" it means it was not installed correctly.

   ```
   quire --help
   ```

   Search or post to our Discussions Forum to troubleshoot installation issues.

Windows Installation

Installing Quire requires using Windows PowerShell command-line shell as an Administrator. Open it by right clicking on Start and selecting "Windows PowerShell (Admin)", or search for it in the search bar and select "Run as Administrator". If you are new to the command-line, read our tutorial *Working in a Command-line Shell*.

Quick Install

If you're eager to get started, this will install the complete Quire package but without e-book or PDF output capability. These may be added later, by following steps 3–5 in the Full Install guidelines below.

1. Download and install Git for Windows: https://gitforwindows.org/
2. Download and install the **LTS** version of Node.js: https://nodejs.org
3. In Windows PowerShell (Admin), install Windows Build Tools with: `npm install --g --production windows-build-tools`
4. In Windows PowerShell (Admin), install Quire with: `npm install --global @thegetty/quire-cli`
5. Confirm installation by pulling up a list of Quire commands: `quire --help`

The Quire installation process may take a minute or two, during which time there will be messaging output in PowerShell. The only messages of any concern are those labeled as ERROR or ERR. If you see these errors, or if you see "command not found" after entering `quire --help` in step 5., search our Discussions Forum to troubleshoot installation issues.

Full Install

1. **Git for Windows** installs useful version control software on your computer. Download the EXE installer file at https://gitforwindows

.org, click on it and hit "run", you should see a setup wizard screen that will install Git for Windows. During the installation, use the default settings.

2. **Node.js** enables you to run javascript on your computer. Visit the Node.js site, and download and install the older LTS (long-term support) version, which is sufficient and more stable than the higher "Current" version that is also available for download. The MSI installer will be downloaded, open it and a setup wizard screen will guide you through the process: https://nodejs.org.

3. **Pandoc** serves two purposes in Quire: You can use it to convert Word documents to Markdown, and it enables you to create EPUB e-book files of your Quire project with the `quire epub` command. Download the Pandoc MSI installer file, open it and a setup wizard screen will guide you through the process: https://pandoc.org/installing.html

4. **Kindle Previewer**, along with Pandoc, enables you to create MOBI e-book files of your Quire project with the `quire mobi` command. Visit Amazon's Kindle Previewer page and download the Widows version: https://kdp.amazon.com/en_US/help/topic/G202131170. Install by double clicking the icon and following the prompts.

5. **PrinceXML** enables you to create a PDF version of your Quire project. At https://www.princexml.com/download/, download either the 32-bit or 64-bit EXE installer depending on your operating system. (If you're not sure of your system open the Settings app by pressing Windows+I. Go to System > About and look for the "System type" entry on the right side.) Click on the downloaded file and hit "run", you should see a setup wizard screen that will guide you through install.

6. **Windows Build Tools** is a set of developer tools for your PC. To install them, open PowerShell as administrator, type the following command, and hit enter:

```
npm install --g --production windows-build-tools
```

This command installs c++ 2015 build tools and python 2 required for node-gyp. The process will take some time and you'll see the starting prompt with the name of your computer once it's complete.

7. **Quire** is operated through a command-line interface (CLI) that enables you to create, preview and output publications using Windows PowerShell with commands like `quire new`, `quire preview` and `quire site`. You can learn more in the *Quire Commands* chapter of our guide. Copy and paste the following line into PowerShell to install Quire:

```
npm install --global @thegetty/quire-cli
```

The Quire installation process may take a minute or two, during which time there will be messaging output in PowerShell. The only messages of any concern are those labeled as ERROR or ERR. These likely indicate a failed installation.

To verify installation, enter the command below. This will give you a list of commands that will help you get started using and navigating Quire. And if you get "command not found" it means it was not installed correctly.

```
quire --help
```

Search or post to our Discussions Forum to troubleshoot installation issues.

Update Quire

WARNING!

Updating Quire will not effect your existing projects. It will only take effect on any new projects you start after updating.

To update Quire to the latest version, you can run the single install command again. In your command-line shell (Terminal on macOS and PowerShell in Administrator mode on Windows), type the following text and hit enter:

```
npm install --global @thegetty/quire-cli
```

Note that while any *new* projects you start at this point will use the latest version of Quire just installed, older, previously started projects will remain untouched. This is to avoid any unforeseen and unintended changes to publications that are completed or in progress as the version of Quire you use can effect layouts of a page, availability of features, etcetera.

While it will not make changes to older projects, you should still be able to run commands from the new version of Quire on old Quire projects. Commands like `quire preview` and `quire pdf`. If you run into trouble with this and find commands do not run as intended or you see errors logged in your command-line, search or post to our Discussions Forum for help.

Update an Existing Project

If you have an existing project that you want to migrate to a newer version of Quire, we recommend the following process:

1. Update Quire following the instructions above.

2. Start a new quire project.
3. Delete the `content`, `data`, `static` and `layouts` folders in the new project, and replace them with those from your old project.
4. Copy over any style customization changes you made in the `themes/default/layouts/source/css/variables.scss` file or others. (In older versions of Quire this file would be `themes/quire-starter-theme/layouts/source/css/variables.scss`.)

Run `quire preview --verbose` to test. If you see errors logged in the command-line, or if the preview doesn't display as intended on http://localhost:1313/, search our Discussions Forum for help.

Uninstall Quire

In your command-line shell (Terminal on macOS and PowerShell in Administrator mode on Windows), type the following text and hit enter:

```
npm uninstall --global @thegetty/quire-cli
```

This will only uninstall Quire itself. External software you installed as part of Quire (including Node.js, Pandoc, Kindle Previewer, and PrinceXML) will need to be uninstalled individually according to the directions they provide.

Get Started

Dive in to creating your first Quire publication

Before getting started, if you have not done so already, we recommend taking some time to review our beginner's guide *Tutorial: Quire Basics* which is step-by-step introduction to the fundamentals of Quire.

Start a New Project

Now that you have taken the tutorial and installed Quire its time to get started on your first publication. To create a new project you will be running the `quire new` command. You can learn more about various commands in the *Quire Commands* chapter of our guide. Open your command-line shell and copy and paste the text below, replacing `project-name` with what you would like your project folder to be called. (Don't use spaces or special characters in your project name, and lowercase is recommended.)

```
quire new project-name
```

Quire will download a new starter project into a folder named "my-project" in your home directory. The process may take a minute as Quire installs a new starter project (a sample publication including content, images, and

relevant metadata that you can build off of) into a folder named `my-project` in your home directory.

The project is ready when you see the message: "Theme and dependencies successfully installed."

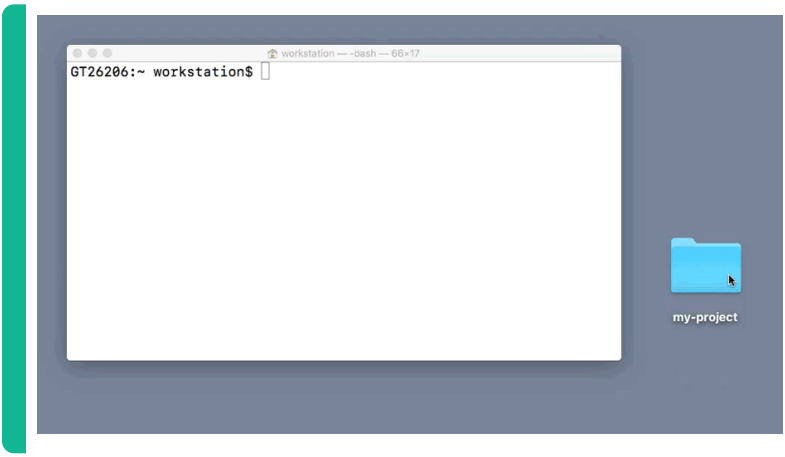
Copy an Existing Project

In addition to starting a Quire project from scratch as described in the previous section, you can also copy and work on a pre-existing Quire project. You would do this if you were on a team working on a publication together and are sharing the files via GitHub or another service, or if you wanted to use a previous Quire project as a template for a new one.

1. Copy the Quire project directory into your home directory (typically from a thumb drive, Dropbox or Google Drive, or GitHub).
2. Open your command-line shell and navigate to the project directory using the `cd` (change directory) command. For example, if your project directory was called `my-project` and it was in your home directory, you would enter `cd my-project`.
3. Still in the command-line shell, type `quire install` and press enter to install the theme dependencies for your project. (This is done automatically when running `quire new`, but needs to be done manually when working on pre-existing projects.)

TIP!

You can also type `cd` and a space in your shell and then drag and drop the Quire directory icon into it. This will copy the full file path.



Files for Content Creators and Editors

Inside each Quire project, you will find the following directories and files. Content creators and editors will primarily use the `content`, `data`, and `static` directories.

```
bin
config
config.yml
content     <-- Markdown files with publication text.
data        <-- YAML files with publication data.
README.md
site
static      <-- Images / Style overrides / PDF, EPUB & MOBI
themes          files that are output with `quire pdf` etc.
```

content

The central part of Quire is the `content` directory where almost all of a publication's text content will live as individual Markdown files. Every

Markdown file is a *page* of the publication. You can read more about how to structure the publication content in *Pages*.

> **TIP!**
>
> **New Quire projects started with the `quire new` command come with some demo content, images, and data as samples to start. These materials can be written over, re-used, or deleted altogether as you'd like.**

data

What content doesn't live in `content` directory as a Markdown file, will live here in the `data` directory as a YAML file. A `publication.yml` file is required (read more in *Publication Metadata & Configuration*), but a Quire project may also include `references.yml` (*Citations & Bibliographies*); `figures.yml` (*Figures*); and `objects.yml` (*Catalogue Objects*).

static

The `static` directory includes anything that will be included in your final publication, but that doesn't have to first be processed through Quire's static site generator. By default, this includes a `css` directory for directly overriding theme styles (read more in *Customizing Styles*); a `downloads` directory for the multiple Quire formats (*Output Your Project*); and an `img` directory for all image and other media assets (*Figure Images*).

README.md

The `README.md` file is a code convention, and is a free space for information about the publication. **It is not used in the output Quire publication at all.** However, if you host your Quire project on GitHub or other similar `git` project management sites, the `README.md` file is used for the repository's front page description. Often it will include notes on development, on what usage is allowed, on how issues will be handled, and if contributions should be considered.

Files for Developers

Inside each Quire project, you will find the following directories and files. Developers will primarily use the `config.yml` file and the `bin`, `config`, `site`, and `theme` directories.

```
bin          <-- Scripts
config       <-- Secondary/environmental configuration
config.yml   <-- Main configuration
content
data
README.md
site         <-- The built site output with `quire site`
static
themes       <-- Layouts, shortcodes, styles, js
```

bin

Currently, it only contains a `deploy.sh` script file for deploying a Quire project to GitHub pages, which you can learn more about in the *Deploy Your Project*.

config.yml

This is a standard, required file for Hugo and also for Quire. In Quire, it is used expressly for configuring how Hugo operates, and for defining a number of key values used in Quire templates. Users who have worked on other non-Quire/Hugo projects will note that they typically use the `config.yml` file to also store publication metadata. Given the potentially large scope of this metadata in formal digital publications, Quire uses the `publication.yml` file inside the `data` directory instead. Read more in *Publication Metadata & Configuration*.

config

This is an additional configuration directory. While most Quire configuration happens in the `config.yml` file as explained above, the

`config` directory gives more specific controls for different output formats and development environments. In most cases, changes won't need to be made to these files until you are deploying your site. Read more in *Output Your Project*.

📁 site

This is where the built pages of the Quire website will live. This folder and its contents are automatically generated with the `quire site` command, and should not be edited directly. Read more in *Output Your Project*.

📁 themes

The `themes` directory contains one or more themes that define the structure and style of the Quire publication. When using the `quire new` command, the theme is `default`. Read more in *Customizing Styles*.

Create a Publication Outline

It is a good idea to start any project by creating a basic outline of your publication. To get started with your outline, you will want to download a text editor. We recommend Atom or Visual Studio Code, two free and fully featured options. Once the text editor has been installed, open your Quire project in it. You will see the directory contents listed on the left sidebar. The way you organize the Markdown files in the `content` directory of your project will define the structure of your publication and how the *Table of Contents* is organized.

Here's an outline showing the order, organization, and file names for a sample publication:

```
cover.md
contents.md
part-one
  section-overview.md
```

```
    chapter-01.md
    chapter-02.md
  part-two
    section-overview.md
    chapter-03.md
```

The names of the files will effect the final URLs of your publication. By default, URLs will be the filename, minus the `.md` suffix. Files nested in a sub-directory within `content` will include that sub-directory in the URL as well.

File	URL
The `cover.md` file	`mypublication.com/cover/`
The `contents.md` file	`mypublication.com/contents/`
The `chapter-01.md` file inside the `part-one` directory	`mypublication.com/part-one/chapter-01/`
The `section-overview.md` file inside the `part-two` directory	`mypublication.com/part-two/section-overview/`

TIP!

To have URLs for your homepage or section landing pages that don't include the Markdown file name, add `slug: .` to the page YAML of that file. Read more in the *Pages* section of this guide.

For the ordering of the pages, in the example above we've listed the files and directories as they would appear in the publication's table of contents. When looking in the actual `content` directory on your computer or in your text editor, however, they will almost certainly not appear in the

proper publication order. More likely, they'll appear alphabetically or by date modified, which is also how Quire will order them when building and previewing your publication. You can adjust this by assigning a `weight` to each page in its page YAML.

There are some other important rules and tips to keep in mind:

1. **To create a new file, select "New File" in the text editor menu.** You can also right click or press control click on a folder and select "New File".

2. **Filenames should be lowercase, with no spaces. Always include the .md suffix.** If file names contain more than one word, use a hyphen to separate them. Make sure that the file name includes .md.

3. **Sub-directories can't have other sub-directories within them.** Quire currently supports only one level of nesting.

4. **Don't use `index.md` or `_index.md` files.** Though common for users with previous static site or web development experience, you should not use `index.md` or `_index.md` files in your Quire project. Because of the way Hugo is modeled, these work against the linear ordering of the publication and break the *Next* and *Previous* page navigation in Quire.

Prepare Images and Text

TK

Preview and Edit a Project

Quire lets you preview the current version of your site in a web browser, and will update the preview as you edit the files.

To run the preview:

1. Open your command-line shell and navigate to your Quire project directory using the `cd` (change directory) command. For example,

if your project directory was called `my-project` and it was in your home directory, you'd enter `cd my-project`.

2. Still in the command-line shell, type `quire preview` and press enter to start the preview server.

3. Open a web browser and visit http://localhost:1313 to see the publication. To stop the preview you can either press Control–C or type `quire stop` and press enter.

Some tips for previewing your publication outline:

1. **Include YAML on page for it to be viewable in your web browser** In order for pages to become active, you must have basic YAML included at the top of the page. Learn more about YAML in *Markdown & YAML*

2. **Use menu:false to hide a page from the table of contents view.** If you want to hide a page from the table of contents include `menu:false` in the YAML.

> **WARNING!**
>
> **In some cases, changes to `.yml`, `.scss` and `.css` files may not show up in your preview immediately. You may need to refresh the browser, clear the browser cache, or stop and re-start the `quire preview` command in these cases.**

Project Management with GitHub

Use GitHub to host your project, track changes, and collaborate

GitHub is a project management platform centered around git-based version control. While it was originally developed as a useful tool for developers, it is now used by artists, writers, and other non-tech professionals. In fact, Getty uses GitHub to manage all of its publications, as well as Quire itself. We strongly recommend using GitHub as a tool for managing your Quire projects (whether collaborating or working solo).

Here are some of GitHub's main advantages:

- GitHub is free to use.
- It's a secure platform for hosting your project's code and creating an electronic backup of your work.
- GitHub saves the entire version history of your project (including deleted files), enabling you to track changes, revert changes, and restore earlier versions.
- You can use GitHub to create test versions of your project and experiment with new ideas before officially incorporating them.

- Github makes working with collaborators easy. You and your colleagues can work in different versions of the project, review and comment on one another's work, and consolidate any approved changes into the master project.
- In the case of Quire, we also use GitHub to host our community forum where you can ask and answer questions, share ideas, and assist fellow Quire community members.

> **TIP!**
>
> **For more information, we recommend checking out GitHub Docs to learn by topic, GitHub Glossary to understand key terms, and GitHub Guides for a broad overview. We also encourage you to check out Coding Train's Git and Github for Poets video series as a helpful primer.**

To get started:

1. Sign up for a free GitHub account
2. Download and install GitHub Desktop, unless you're familiar with using git from the command-line
3. Download and install Git LFS (Large File Storage). Learn more in the Manage Large File Sizes with Git LFS section below.

Next, you will have two options:

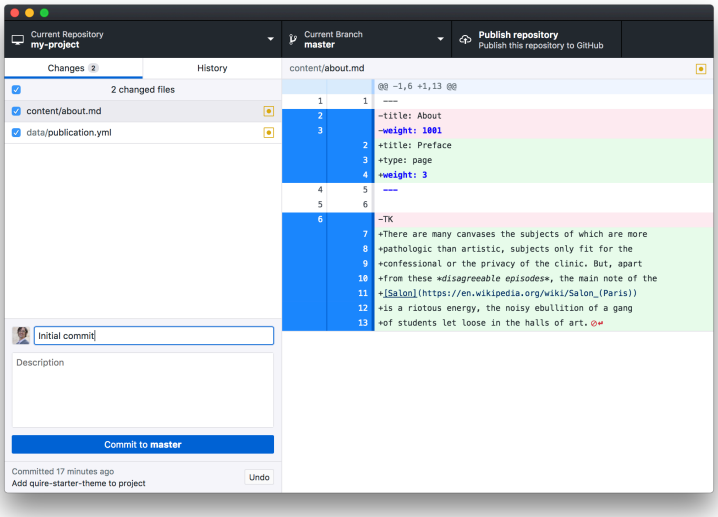
1. **Host Your Project's Code on GitHub**: Create a new repository on GitHub and add a project that lives locally on your computer. (This would be the case if you recently used `quire new`.) Please note, you are only hosting your project's code and not the actual online version of it.
2. **Clone an Existing Project from Github**: Copy an existing repository off GitHub to work locally on your computer. This would be the case

when collaborating with others who have already created a repository.

Host Your Project's Code on GitHub

GitHub uses something called "git" to track changes in project files. When you start a Quire project with `quire new`, it comes already configured as a git repository. (You might have noticed a mysterious `.git` directory in your folder.) This means your project is ready to be uploaded to GitHub as a new repository.

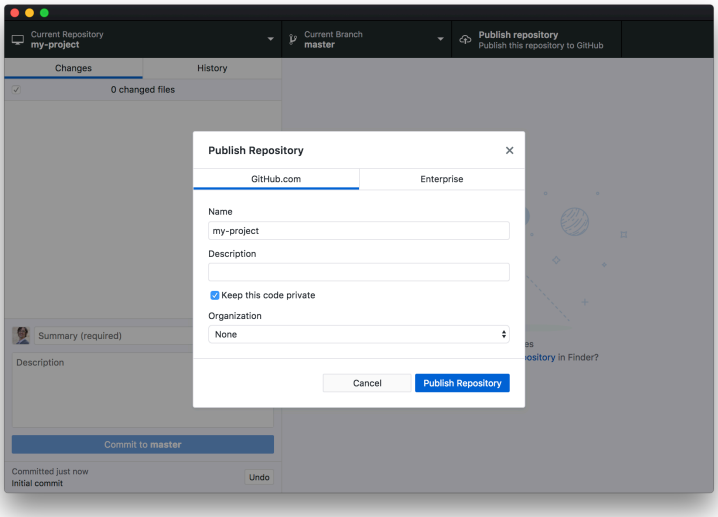
1. Open GitHub Desktop and click the upper left-hand corner where it says "Current Repository." Click "Add" and choose the option to "Add Existing repository..."
2. Navigate to the project in your home directory and click "Add Repository."
3. If there are any items listed in the window's left-hand pane, you will need to commit them to your project. As this will be your first commit, type "Initial commit" in the Summary field and click the "Commit to master" button.



In GitHub Desktop, files with changes will be listed at left, and a detailed red (old version) and green (new version) highlighted "diff" of individual changes will appear at right. The Commit to Master button in the lower left adds these changes to your project's git repository.

---

4. Click the blue-button that says "Publish Repository." This will push your repository up to GitHub.com. In the pop-window, keep the name as is and add a description if you would like. Then click "Publish Repository."



When publishing a new project repository to GitHub leave the name the same as your project file. You can add add a description if you'd like, can choose whether to make it private or not if you have a paid account, or publish it to an organization account if you have publishing access to one.

Your project code is now on GitHub! To see it, go to Repository > View on GitHub or visit https://github.com/YOUR-USERNAME/YOUR-PROJECT-NAME.

Clone an Existing Project from GitHub

If you are collaborating on a project, and one of your colleagues has already hosted your project code on GitHub, you can clone (or make a copy) of this repository to work on locally.

1. Click on the upper left-hand corner of GitHub Desktop where it says "Current Repository." Click "Add" and choose the option to "Clone repository..."

2. Click the URL menu option and cut and paste the URL for your desired repository. Be sure to choose the local path where you want

your project to live on your computer. We recommend creating a GitHub folder in your home directory and storing the project in there. (Alternately, you can copy this project directly from GitHub.com. Navigate to the project on GitHub, click the green "Code" button, select "Open with GitHub Desktop," and follow the prompts.)

3. You now have a copy of that project on your computer connected to the original code on GitHub. Before you can collaborate, you will need to either add your colleagues as collaborators or host your project in an organizational GitHub account where everyone is a member. Learn more in the *Collaborate with Others Using GitHub* section below.

> **WARNING!**
>
> **To view a preview of a Quire project cloned from GitHub, you'll need to first run the `quire install` command in your command-line shell. Then run `quire preview` and navigate to http://localhost:1313/.**

Manage Project Changes on GitHub

There are three basic steps for making and managing changes to your project on GitHub:

1. Save changes in your text editor
2. Commit those changes to the git repository
3. Push those commits up to GitHub

Save Changes

Open your project in your text editor (File > Open > [your project name]), make, and save changes. When you go into GitHub Desktop, you should

see a list of all the files added, deleted, or edited in the left-hand pane. On the right, you'll see exactly what changes were made inside each file. Old text will be highlighted in red, while any new text will be highlighted in green. This enables you to compare and review changes.

TIP!

Save changes with File > Save (or Command-S on Macs and Control-S on PCs). If working on multiple markdown files, make sure to save each file individually. You will see a little blue dot at the top of the page if there are any unsaved changes.

Commit Changes

At this point, your changes have been saved locally (i.e. on your computer), but they aren't in the git record of your project yet. For that, you need to make a commit, a small grouping of meaningful changes.

To make a commit, at the bottom left, above where it says description, write a short name summarizing the change you're making (a longer description is optional) and click the "Commit to master" button. Keep in mind, a key advantage of GitHub is that you can undo your work by reverting commits and returning to earlier versions of your project. Here are a few tips for styling your commits to make that process easier:

- Keep the commit name concise (around 50 characters max), so you can quickly and easily browse them in the future.
- If you need to explain the commit in greater detail, use the description field (around 72 characters max).
- The summary should be a phrase that completes the sentence "This commit will __."
- Use present/imperative tense, such as "Add contributor names and bios;" "Fix typos in section pages;" or "Change header color."

When naming commits, think of them as purposefully connected bundles that you could undo at a later date without affecting other areas of your project. Rather than committing all the random changes you made to a single file or everything you did before lunch that day, make small and frequent commits. And remember to commit changes that *make sense together*.

TIP!

Changes are organized by the file in which they were made. Keep this in mind if making multiple types of changes to one file. Use the checkboxes at the lefthand side of your GitHub Desktop window to select which changes will or won't be included in a given commit. Break your work into individual commits or commits with multiple changes.

Push Changes

Even after saving changes in your text editor and committing them in GitHub Desktop, GitHub still doesn't know about them. For that, you need to push your commits to GitHub.

Click on the "Push origin" link in the upper-right corner. (If you haven't committed changes yet, this area will say "Fetch Origin" instead). Pushing can include one or many commits, and you can push as often as you'd like. It's typically good to do it at least a couple of times a day when actively working on a project.

Now the changes to your project have been successfully saved in GitHub!

Collaborate with Others Using GitHub

One of the main advantages of using GitHub is that it's an excellent tool for managing projects with multiple people working on them. Once the project is on GitHub and cloned by you and your colleagues, you can all make changes and commits to the same project.

When setting-up a new repository you have two options: you can invite users to become collaborators to your *personal repository* or groups of people can collaborate across many projects simultaneously with *organization accounts*. Each has different options for managing settings and permissions.

Clear communication is vital when collaborating with GitHub. If multiple people are working on a project and all are making changes to the same files, there's a good chance you will run into conflicts. While git is specifically designed to deal with these sorts of issues, it's important to avoid working on the same sections at the same time.

Create a Branch

A good way to manage users' changes and avoid conflicts is by using branches. Branches are essentially parallel versions of a project. Users can make changes on their branch without affecting the main version of the project. This allows you to review changes and identify possible conflicts before they are incorporated into your project.

1. Before starting a new branch, click the "Fetch Origin" button at the top right. This will pull down any recently merged changes to the master project. That way, when you start a new branch, you are working off the most up-to-date version of your project.

2. Navigate to where it says "Current Branch" at the top of your GitHub Desktop window.

3. Click the drop-down arrow and select "new branch" to the right of the Filter box.

4. Name your branch.

5. Select "main" where it says "Create branch based on." This means you are creating a new branch that runs parallel to the main project. Every time you open the project to work on it, make sure you are working in the correct branch.

Tips for naming branches:

- Keep branch names concise.
- A branch name should be indicative of the work you are doing or who is working on that particular branch.
- Do not use uppercase letters.
- Separate words with dashes (GitHub won't permit spaces).

Submit a Pull Request

Once you have pushed the changes on your branch, you will be prompted to submit a pull request. This is a request for your collaborators to review and comment on changes.

1. Click the blue-button that says "Create Pull Request." This will take you to GitHub.com.
2. Confirm that the branch in the "base" drop-down menu is where you want to merge your changes. In most cases, this will be the master or main branch.
3. You can leave the default title or create a new title and description for your pull request.
4. On the right-hand side, you will see an option to choose the reviewer(s) you would like to overlook your changes.
5. If all your changes are ready for review, click the green-button that says "Create Pull Request." Or click the drop-down arrow and choose "Create Draft Pull Request" if you are still making changes in this particular pull request.

Review & Merge Changes

Now you have the opportunity to review changes, provide feedback, or request edits. Once the pull request has been approved, you can merge these changes into the master version of your project.

1. Navigate to “Pull Requests” on the main page of your repository.
2. In the list of pull requests, choose the one you’d like to review.
3. Click “+/- Files changed”.
4. Hover over the section of text you would like to comment on.
5. You will see a blue square with a plus mark pop-up. Write your comments here.
6. When you are done commenting click, “Start a Review.” You will have the option to comment, approve, and request changes.
7. Once the pull request has been approved, provided there are no conflicts, you will see a green-button that says “merge pull request.” Click this and, voila, your changes are not incorporated.

Merge conflicts will happen when you attempt to merge branches that have competing commits. Read more about addressing and resolving merge conflicts in the GitHub documentation.

Manage Third Party Assets with GitHub

If you plan to make your repository visible to the public at some point, **you should not commit third-party licensed assets into it**, as this can expose those assets to easy, unlicensed use by other people. We recommend either keeping those assets out of the git record completely and managing them locally or adding them in as a secondary repository that always

remains private but can be connected to the main repo as a git submodule.

Keeping Assets Out of Git

For anything you do not want to include in git, you can add a listing for it in your project's main `.gitignore` file (which you can see when viewing your project in your text editor). You can choose to ignore files by name or by extension, or you can ignore entire directories.

For instance, you might add all your third-party licensed images into a `licensed` folder in your `static/img/` folder. By adding `static/img/licensed/` to a new line in your `.gitignore` file, none of those files will be added to the git record and uploaded to GitHub. You'll only have access to ignored files locally or for collaborators if you provide them copies of the files that they can manually add into their own copies of the repository.

> **WARNING!**
>
> **Items added to your `.gitignore` file that have already been committed to your repository will no longer be tracked going forward but will still exist in your project's history. Permanently removing files is possible, but much more of a process.**

Keeping Assets in a Git Submodule

Where using `.gitignore` can ensure certain sensitive or third-party files won't get added into a project repository and thus be uploaded to GitHub, it means you have to manually track and manage those files yourself. Another option is to create a second project repository on GitHub to stay private even while your main project is made public. You include that second repository as a submodule within your main repository.

For instance, you might add all your third-party licensed images into a `my-project-licensed-images` repository and then link that repository as a submodule into the `static/img/licensed/` folder of your main project repository. In this way, Quire can still build the site and output files and preview those images as normal. The files will also be under version control with git and connected to your project (so there's no manual syncing to do), but you'll be able to keep them private to everyone except your team or collaborators.

There are good directions on working with submodules on GitHub's blog. Note, however that managing commits to a submodule repository within a repository can be a little tricky, so it's best not to go this route unless you are comfortable with git and GitHub already or have access to good support.

Manage Large File Sizes with Git LFS

Quire projects are pre-configured to use Git LFS (Large File Storage) to manage all the files in your project's `static/downloads/` and `static/img/` directories. This is to avoid running into issues with large individual files (GitHub blocks files greater than 100MB) and to keep the overall repository size down to a manageable level. File size issues can be particularly common when it comes to your project's various output formats, including the PDF, EPUB, and MOBI.

Git LFS is a tool that moves your large files elsewhere and in their place stores references pointing to them in your repo. GitHub then uses these references as a guide to locate your large files. To start, download and install Git LFS (Large File Storage) if you haven't already done so.

When adding your new Quire project repository to GitHub Desktop, you'll see a message that says: "This repository uses Git LFS. To contribute to it, Git LFS must first be initialized." You can click the "Initialize Git LFS" button, and then all files will be tracked and stored using LFS.

GitHub accounts currently include 1GB of free Git LFS storage and bandwidth usage, and more can be purchased as necessary. An additional 50GB costs $5/month.

If you do not want to use Git LFS, delete the `.gitattributes` file from your project. This file can also be updated manually or by using the `git lfs track` command from the command line if you want to change or add to what files or folders are being tracked.

Quire Commands

Learn how to start and preview projects, output files, and more

Quire commands are typically run from Terminal on a Mac and Git Bash (or its equivalent) on a PC. They control creating, previewing, and outputting Quire projects. The following commands are available.

TIP!

Run `quire --help` in your command-line shell for a full list of the Quire commands and options defined below.

Start and Preview Projects

`quire new project-name`

Create a new Quire project named `project-name` in the current directory. The name can be anything, but shouldn't contain spaces or special characters.

`quire preview`

Run a local server to preview the project in a browser. Defaults to previewing at http://localhost:1313/, but will use other port numbers (such as http://localhost:6532/) if 1313 is busy. The specific address will be listed in your command-line terminal after running the

command. If you're having any issue with the preview, try running `quire preview --verbose` instead. This outputs error, warning, and other processing information that can sometimes be useful in troubleshooting.

`quire install`
Install this project's theme dependencies when you update or change themes.

`quire stop`
Stop the preview from running. This can also be done by typing Control-C.

Output Files

`quire site`
Build the final web files for your publication into its `site` directory. These include all the pages, images, and styles necessary for your project, and can be hosted on any web server.

`quire pdf`
Generate a PDF version of your publication into its `static/downloads/` directory as `output.pdf`.

`quire epub`
Generate an EPUB version of your publication into its `static/downloads/` directory as `output.epub`.

`quire mobi`
Generate an MOBI (Kindle) version of your publication into its `static/downloads/` directory as `output.mobi`.

◆ ◆ ◆

For the PDF, EPUB, and MOBI commands you can specify a new filename with the `--file` option.

```
quire pdf --file=photography
```

Outputs as: `static/downloads/photography.pdf`

Or you can specify a name and an alternative file path. If the directories don't already exist in your project, Quire will create them and output the file there. We recommend always outputting somewhere into the `static` directory as this will automatically be included in your project when you run `quire site` to output the final web files.

```
quire epub --file=static/ebooks/photography
```

Outputs as: `static/ebooks/photography.epub` .

◆ ◆ ◆

Also for the PDF, EPUB, and MOBI commands, developers may use the `--env` option to specify and environmental variable.

Customize File Templates

`quire template epub`
Download the built-in template for customization of the cover and title pages of the EPUB and MOBI files Quire outputs. `epub/template.xhtml` All other template customization (for the website and pdf/print versions) is done in the `theme` directory.

Get Help

`quire -V` **`quire --version`**
Output the version number.

`quire -h` **`quire --help`**
Output usage information.

`quire preview --verbose`
Show verbose output in the command-line. Includes warnings, errors, and process information.

`quire debug`

Development use only. Log info about current project.

YAML & Markdown

Understand the basics of working in plain-text format

Content is stored in two different plain-text formats in Quire: YAML (*yam-ul*) for data, and Markdown for more narrative or textual content. Markdown is used in standalone `.md` files in the `content` directory of every Quire project. YAML is found in `.yml` files in the `data` directory, in the configuration files, and at the top of every Markdown file.

YAML Basics

YAML is designed to be a plain-text way of capturing data. The general principal is to have the name of a data item (a key), followed by a colon, a space, and then the data item's value. A key-value pair.

```
title: My Book
```

Each line in YAML is a new item. Dashes represent individual items in a list. In the example below, there are two contributors, each with a first and last name.

```
item:
other_item:
multiple_items:
  - item_name:
    item_description:
  - item_name:
    item_description:
```

Note that indentations matter in YAML. If any of the items above were indented even just one space more or less from where they are, the YAML would not be formatted correctly, and the Quire preview and output functions would not work. YAML items and list items should always line up with one another.

WARNING!

Improperly formatted YAML can temporarily break Quire functionality. Copy and paste your YAML blocks into a validator like the Code Beautify YAML validator to make sure there aren't any hidden errors.

YAML can include multiple, markdown-style paragraphs by using a pipe character, dropping down a line, and indenting one level. This can be used in areas like captions, descriptions, and abstracts.

```
item: |
  Using a pipe character, and then dropping down a line
  and indenting like this allows you to include multiple
  paragraphs, just as you would in Markdown.
```

```
  Not all Quire YAML attributes expect Markdown though,
  so check the docs.

  - Markdown style lists
  - and other formatting are
  - also allowed
```

YAML block entries can be in any order. It doesn't matter if you write:

```
---
title: Cheatsheet
type: page
---
```

Or:

```
---
type: page
title: Cheatsheet
---
```

Certain formatting and characters (like colons within the text, or lines leading off with asterisks meant to italicize some of the text) can cause issues. In the example above, `title: My Chapter` without *My Chapter* in quotes works just fine, but more complicated cases might arise. In these cases, double quotes will help to avoid issues.

```
description: "*My Chapter* is about colons :)"
```

Anything can go within double-quotes, except for other double-quotes. If you need double-quotes, use “curly quotes”, or use a backslash to escape the double quote `\"`.

```
title: "“Ah ha!” Amazing Double-quote (\") Tricks!"
```

TIP!

Our Top 3 YAML Tips:

1. **Use quotes around item values**
2. **Watch horizontal spacing to make sure things line up**
3. **Check your YAML with a validator**

Markdown Basics

Markdown is designed to be a simple, plain-text markup language that uses a few text rules to structure content for easy conversion into HTML. Writing in Markdown should be thought of as giving your content structure, not style. You use Markdown to indicate what’s a heading, what’s a list, etcetera. Quire’s themes and stylesheets then control what those headings, lists, and other elements *look* like, from device to device and format to format.

Special characters like en- and em-dashes, and diacritics work fine in Markdown and in Quire publications. Any Unicode character is allowed. The only limitation, for less common characters, is whether the font you’re using includes it. When a font does not include a specific character, most browsers will substitute one from a different font.

The following sections detail the most commonly used Markdown tags.

Paragraphs

Individual paragraphs are created with double line breaks.

```
This is the first paragraph.

This is the second.
```

This is the first paragraph.

This is the second.

Headings

Headings are created with hashmarks. The number of hashmarks corresponds to the level of the heading you want.

```
### Heading 3
#### Heading 4
##### Heading 5
```

Heading 3

Heading 4

Heading 5

Start your content headings with Heading 2 tags rather than Heading 1. Heading 1 should be reserved for the page title and will be automatically generated in Quire. And, for truly accessible documents, headings should

be thought of as levels of your content outline, not as sizes large to small. See our *Accessibility Principals* for more on this.

Italic and Bold

Italics and bold are created with asterisks.

```
*Italic Text*
**Bold Text**
```

Italic Text

Bold Text

Blockquotes

Blockquotes (indented blocks of text) are created with the right caret, or greater-than sign.

```
> Blockquote
```

> *Blockquote*

Links

Links are created with text in brackets followed immediately by a url in parentheses.

```
[Link Text](http://www.linkadress.com)
```

Link Text

Lists

Dashes and numbers create lists. Indenting creates nested lists.

```
- dashes
- make
  - a list
  - with
  - bullets
```

- dashes
- make
  - a list
  - with
  - bullets

```
1. numbers make
2. a list with
3. numbers
```

1. numbers make
2. a list with
3. numbers

Footnotes

Precede footnote numbers with an up-arrow accent (`^`) and then surround it in square brackets. Footnote number one would be `[^1]`, number two would be `[^2]`, and so on.

At the end of the page, usually under a “Notes” heading, add the corresponding note with the same marker followed by a colon and the note text.

```
## Notes

[^1]: The footnote itself is the same thing as the footnote
number reference in the text, but with a colon followed by
the footnote text
```

Footnotes can also include Markdown formatting, including lists and even multiple paragraphs. For these, indent the content inwards two levels and put a line space in between the paragraphs just as you would elsewhere.

```
## Notes

[^2]:
    Footnotes with multiple paragraphs

    Are indented in twice, and have line breaks between.

    - Markdown lists
    - work like this in footnotes
    - as well
```

WARNING!

The built-in Markdown processor will automatically renumber footnotes in the order they appear in the text. It will also always put the footnotes at the very end of your content, no matter where you may try to put them.

Markdown Special Cases

Markdown and HTML

You can also use HTML tags in a Markdown file. This can be convenient for adding HTML elements that Markdown doesn't support, or for applying special styling. For instance, by wrapping text with a `<span>` tag with a class in order to add custom styling. (See more about this in the *Styles Customization* chapter of this guide.) Note, however, that you can do the same by wrapping multiple paragraphs of Markdown in `<div>`, `<section>` or other block-level tags. For this, you need the `q-class` shortcode.

TIP!

For the things Markdown can't do, Quire includes number of useful shortcodes. You'll read more about them in other chapters of this guide. A complete list is available in the shortcode reference section.

Fractions, Superscripts, and Subscripts

The fractions 1/4, 1/2, and 3/4, will be automatically converted into proper, Unicode fractions (¼, ½, ¾). Other Unicode fractions can also be used in Markdown directly, though note that not all fonts support the eighths in which case, browsers will render them with a default font. The fractions

are: ¼, ½, ¾, ⅛, ⅜, ⅝, ⅞. Any others would need to be written using superscript and subscript formatting.

While some Markdown processors support superscript and subscript formatting with ^ and ~ characters, the one built into Quire does not. You'll need to use the HTML `<sup>` and `<sub>` tags in your Markdown. For example:

- `19 × 24<sup>3</sup>⁄<sub>16</sub> inches` = 19 × 24$^{3}/_{16}$ inches
- `20<sup>th</sup> Century Sculpture` = 20th Century Sculpture
- `Chrome yellow (PbCrO<sub>4</sub>)` = Chrome yellow ($PbCrO_4$)

You will see a `fractions` attribute with a value of "false" in the `config.yml` file of your publication. Changing this to true will automatically render fraction-style superscript and subscript formatting for anything written as an integer followed by a slash and another integer. However, in many instances this will catch things that are not meant to be fractions. For this reason, we recommend leaving `fractions` set to `false`, and manually adding the necessary markup as it's needed.

Markdown Gotchas

1. The built in Markdown processor will incorrectly create links even if there is a space between the bracketed text and the parentheses. For instance, a footnote reference number `[^1]` followed by a space and any text in parentheses, will incorrectly format as a link: `[^1] (Some aside text here)`. To avoid this, you can use the HTML entity reference, `&#40;`, for the first parentheses, or a backslash escape character before the first parentheses.

   ```
   [^1] &#40;Some aside text here)
   [^1] \(Some aside text here)
   ```

2. (c) will automatically render as ©.

Markdown Preview

Many text editors offer a preview function for Markdown, either pre-installed or as an add-on. In Atom for instance, a Markdown file can be previewed by selecting Packages > Markdown Preview > Toggle Preview (or just Shift-Control-M). The preview won't match the style of your publication site, but will have default styling for headings, blockquotes, links and the like to allow you to confirm proper formatting.

Outside of more code-driven text editors, there are also a growing number of Markdown-specific editors. Typora, for instance, offers a single-page live preview by displaying styled Markdown-formatted text as you type it.

Markdown Output Configuration

Hugo has a built-in Markdown processor, Blackfriday, which comes with some configuration options that can be applied in your project's `config.yml` file. Details can be found in the Hugo documentation.

By default, in the `config.yml` file of your Quire project, Blackfriday's `fraction` option has been set to `false` (text that looks like a fraction won't be automatically formatted as such), and the `hrefTargetBlank` option set to `true` (external links will open in new windows/tabs).

Markdown Resources

This guide doesn't cover all existing Markdown tags, but there are some good sources that will help you find the right syntax to format your text. For example, the Programming Historian provides an introductory lesson to Markdown, and John Gruber, the creator of Markdown, provides a comprehensive explanation of the basics and syntax on his personal site Daring Fireball.

Be aware of the multiple Markdown flavors out there and the fact that not all flavors are supported by Blackfriday.

Microsoft Word to Markdown Conversion

Commonly, project content will start from Microsoft Word documents rather than being written originally in Markdown. In these cases, a simple file conversion using Pandoc can be done.

There are some easy things you can do in the Word document prior to conversion to ensure the best possible results:

- We recommend not inserting any images and media into the Word document before conversion.
- Headings should be formatted by applying Word styles instead of by manually changing font formats.
- Don't use any font color or color highlighting, it will not convert to Markdown.
- Save as .docx rather than .doc

While there are a number of free tools, we recommend using Pandoc, which is included with the basic Quire installation and can be used through the command-line. To convert, open your command-line shell, use the `cd` (change directory) command to move to the folder where your .docx documents are saved, and enter the applicable Pandoc command:

To convert a single Word document (in this example it has a file name of MyFile.docx) into Markdown:

```
pandoc --atx-header --wrap=none -s MyFile.docx -t markdown-smart -o MyFile.md
```

To convert all the Word documents in the folder and compile them into a **single** Markdown document:

```
pandoc --atx-header --wrap=none -s *.docx -t markdown-smart -o MyFile.md
```

To convert all the Word documents in the folder into **individual** Markdown files:

```
for f in *.docx; do pandoc --atx-header --wrap=none "$f" -s -t markdown-smart -o
```

Note that the `--atx-header` and `--wrap=none` options in the above commands are optional, but recommended for Quire.

- Quire uses ATX-style Markdown headers, these are specified adding `--atx-header` to the command.
- The lines of a Pandoc-generated file are 80 characters long. Adding the option `--wrap=none` to the command will override the default wrapping making easier to work with your files in the Text editor.

The order of the extensions doesn't matter, and you can either type:

```
pandoc --atx-header --wrap=none -s MyFile.docx -t markdown -o MyFile.md
```

or

```
pandoc -s MyFile.docx -t markdown --atx-header --wrap=none -o MyFilemd
```

Metadata & Configuration

Update crucial information before deploying your publication

Quire uses two YAML files as sources of the metadata and to define how the publication works. In this page, we list the YAML properties and values that need to be defined in the two following files: `config.yml` and `publication.yml`. By default, both `config.yml` and `publication.yml` will be generated when you create a Quire project, however the values of the properties will be either edited or added to the properties listed as we describe below.

You can read more about YAML syntax basics in "YAML & Markdown".

Adjust the Default Publication Settings in config.yml File

The `config.yml` file is a standard and required file for Hugo, and also for Quire. In Quire, it is used expressly for configuring how Hugo operates, and for defining a number of key values used in Quire templates. Users who have worked on other non-Quire Hugo projects will note that they typically use the `config.yml` file to also store publication metadata. Given the potentially large scope of this kind of metadata in formal digital

publications, Quire instead uses the `publication.yml` file inside the `data` directory for that purpose (see below).

The properties in the `config.yml` file are individually documented in the *Developers section*, however, a few key items to note:

- While Quire exclusively uses the `title` value as defined in your `publication.yml` file, other Hugo projects require a `title` value in the `config.yml` file, so it is a good idea to include it here as well.
- The `theme` value should match the name of the folder in the `/themes` directory that contains your theme files; if you've copied the default theme and given it a different name make sure to update the value here too.
- The `params` section includes a number of values specific to various Quire layout templates and shortcodes. All are provided with default values, and should be changed with care. In cases where a value should be deleted entirely, it is usually best to leave it as empty double quotes (`""`) rather than completely deleting it.

Add and Edit Important Metadata in publication.yml File

The `publication.yml` file in the `/data` directory is *the* source of metadata for your publication. While the only value that is truly required is the one for the property `title` , it is a good idea to fill out the `publication.yml` file as completely as possible. Many of the properties are used in the metadata, which is automatically included in the underlying code of every page of the online edition of your publication to support Search Engine Optimization (SEO) and general discovery.

Some key areas are summed up below, and match headings in the `publication.yml` file itself, but there is a detailed documentation of individual properties and their values in the *Developers section* of this guide.

Title & Description

Of the possible properties in this section, `title`, and the optional `subtitle` and `reading_line` are the most important. If your title is particularly long, the `short_title` property can be used to provide an alternative for the navigation elements of the online book where long titles will otherwise be truncated.

It is also a good idea to include both `one_line` and `full` descriptions, as these are used in the publication SEO metadata and often on the *Cover* and *About* or *Copyright* pages.

Publication Details

The values of `url`, `pub_date`, and `language` should be filled out.

- `url` should be the final URL where your publication will live (its permalink) and should include `http://` or `https://` as appropriate.
- The value of `pub_date` must follow a YYYY-MM-DD format (the ISO 8601 format) and should be the projected final publication date.
- Lastly, `language` should be a 2-letter ISO 639-1 language code. The default value is `en` (English) and other languages can be used.

There's an optional `pub_type` property whose values are `book`, `journal-periodical`, or `other`. If you use the value `book`, it is recommended you also include an ISBN as a standard identifier. If you use the value `journal-periodical`, you should include information for the ISSN, `series_periodical_name`, and `series_issue_number` attributes if possible.

Both ISBN and ISSN are considered if you want libraries to catalog your publication. Along with `isbn` and `issn`, `doi` and `uuid` are also supported so you can add these attributes as identifiers:

```
identifier:
  isbn: 978-1-12345-678-9
  uuid: 4a1b423d-6d5a-469b-bd5f-b498182ad6ca
```

DOIs are widely used in academic contexts to support citation while UUIDs serve to identify information in computer systems.

> **TIP!**
>
> **Note that the `isbn` and `issn` identifiers used here are for the online edition specifically. Identifiers for other specific editions (PDF/Print, EPUB, and MOBI) can be defined separately with the appropriate `resource_link`. See the *Formats, Resources & Links* section below for more.**

Lastly, Quire supports publications with multiple publishers, but at least one `publisher` should be listed with a `name`, `location`, and `url` attributes. In particular, this is used in the citation features as well as in search engine metadata.

Contributors

Every publication should have at least one `contributor`. The `contributor` item `type` can have one of three values: `primary`, `secondary`, or `project-team`. The `primary` contributors are those who would show up on the *Cover*, *Menu*, and *Title Page* of a publication, and may include authors, editors, translators, and others. Contributors should, at a minimum, be listed with a `first_name` and `last_name` (or alternately just a `full_name`).

An optional `contributor_as_it_appears` value allows for more fine-grained control in the way contributors are listed. It could be, for example, something like "Edited by Rose Valland and Denis Diderot". Even when

using `contributor_as_it_appears`, the contributors should still be individually listed as contributors (with a value of `primary`) for search engine legibility.

The editors, designers, developers, and others who worked on the title may be listed as contributors with the `project-team` value. This information is usually then listed on the *About* and *Copyright* pages of the publication.

Read more about this matter in the *Contributors* chapter of this guide.

Copyright & License

You should include a `copyright` line property for your publication, and, optionally, `license` information property if you are distributing the publication Open Access.

A simple Copyright statement would typically be formatted as "© 2019 Author Name".

> **TIP!**
>
> **The `copyright` property does support Markdown formatting to allow for multiple paragraphs and other formatting.**

Open access licensing typically means applying one of seven Creative Commons Licenses to your publication. This is in addition to your copyright statement.

Note, an open Creative Commons license does not replace or supersede copyright in a work, it instead says that the copyright holder is licensing (allowing) others to make use of the work in an open way.

To use a Creative Commons license fill in the `name`, `abbreviation`, `url`, and `scope` values of the license property. `scope` value should be either `full`, `text-only`, or `some-exceptions` and will determine the way the license is worded on your site. To override the wording and link language use the `online_text` and `pdf_ebook_text` attributes.

If the `abbreviation` attribute matches one of the seven Creative Commons Licenses, an icon will automatically be included, otherwise you can use the `icon` attribute to point to a specific image file in your images directory.

Formats, Resources, & Links

A publication can have multiple `resource_link` properties, each with the `type` of `other_format`, `related_resource`, or `footer_link`.

- `other_format` will be where you can list the PDF, EPUB, and MOBI editions of your publication that Quire produces.
- `related_resource` are for additional items you want to point readers to.
- `footer_link` are just that and are often links to privacy policies, your own *About* page, or social media profiles.

`resource_link` properties can also be internal pages of the publication or files from your publication, or can point to external resources or other websites. The attributes `type`, `name` (how the resource link will be listed in your publication), and `url` are required.

To facilitate machine readability, it is a good idea to also include `link_relation` and `media_type` attributes from the IANA lists if applicable ones for your particular resource are available.

Subjects

Any number of subjects can be added to the publication in order to aid search engine discoverability. They may be formatted as simple keywords, BISAC Subject Codes, or linked data using the Getty Vocabularies, including AAT, ULAN, and TGN.

For each subject, indicate the `type`: `keyword`, `bisac`, or `getty`. For `keyword`, you only need to include a single comma-separated list under the `name` attribute.

```
subjects:
  - type: keyword
    name: French painting, 19th Century, Delacroix
```

For all others, each subject should be listed individually and should also include an `identifier` attribute. For `bisac` subjects the `identifier` is the BISAC code, for the Getty vocabularies, it's the vocabulary's semantic URL.

```
subjects:
  - type: bisac
    name: "ART / European"
    identifier: ART015030
  - type: getty
    name: "Romantic"
    identifier: http://vocab.getty.edu/aat/300172863
  - type: getty
    name: "Eugène Delacroix"
    identifier: http://vocab.getty.edu/ulan/500115509
```

Revision History

A history of post-publication revisions made to the publication typically appears on the *About* page. Any number of revision history property items can be added and each must include the attributes `date` and a `summary` of changes made on that date. The `summary` attribute supports Markdown formatting, and would typically be in list form.

If you are using GitHub or a similar service for more granular version control, you may also include the `repository_url` in this section. And in

this case the revision history collected in the `publication.yml` can act as an overview. For more, see our revision history policy document.

Page Types & Structure

Define and structure your page contents with YAML

Every page in a Quire publication starts with a block of YAML. The three core attributes you're probably going to define on every page are `title`, `type`, and `weight`. All page YAML, no matter how many attributes it has, goes between a set of three dashes at the very top of the page.

```
---
title:
type:
weight:
---
```

Much more information about the page than just these three attributes can be included. A more complete example would be:

```
---
label:
title:
```

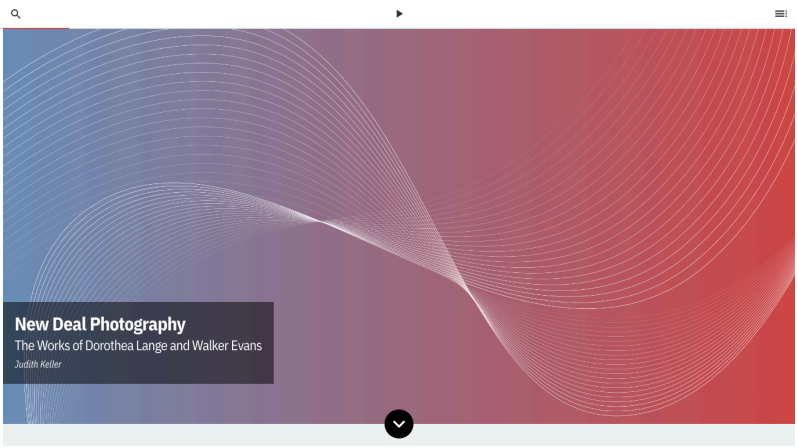
```
subtitle:
short_title:
object:
  - id:
contributor:
  - id:
abstract:
type:
class:
weight:
slug:
---
```

For more details on this full list of possible attributes that Quire can use in page YAML, see the Page API section of the Developer documentation.

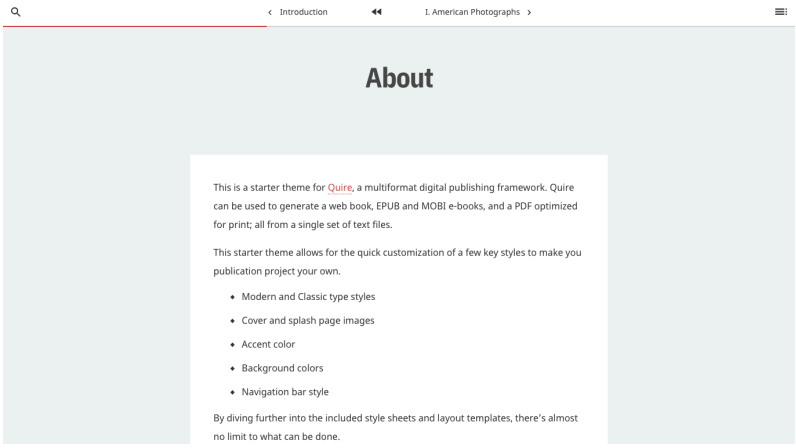
Define Page Types

```
type:
```

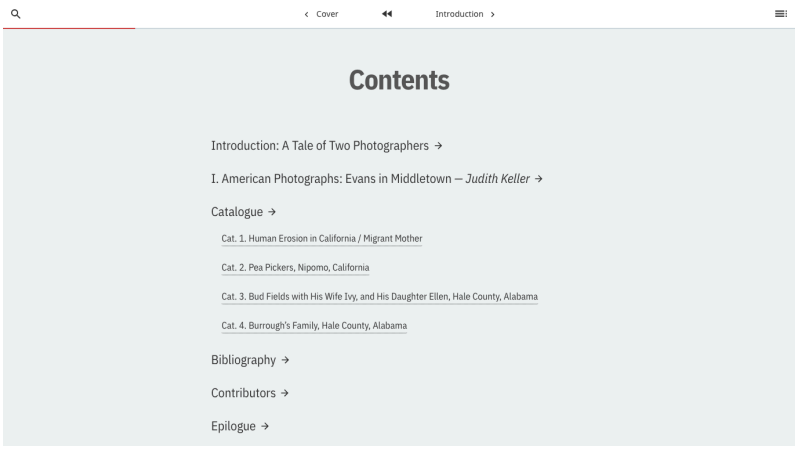
The page `type` must be one of six possible values: `page`, `essay`, `entry`, `cover`, `contents`, or `splash`. If left blank, or if any other value besides these six is entered, the type will default to `page`. (A seventh page type, `data`, is available for special applications such as the pre-built search index. New page types can be created to customize Quire projects even further.)



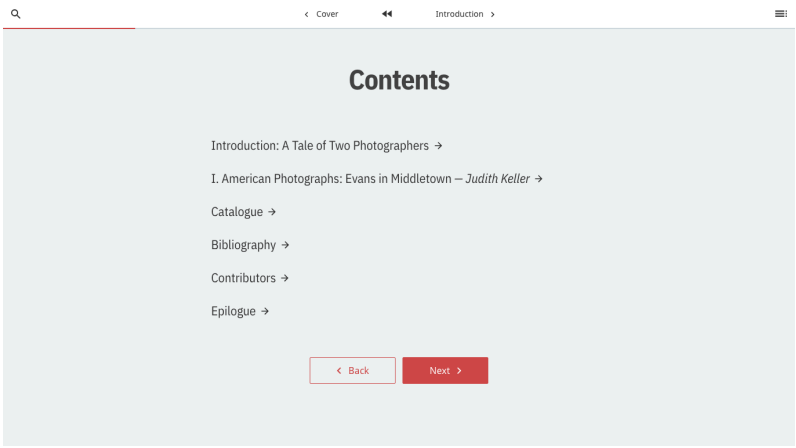
`type: cover`
Cover page in the default modern theme. A custom cover image can be added in the YAML of the cover page: `image: my-cover-image.jpg` .



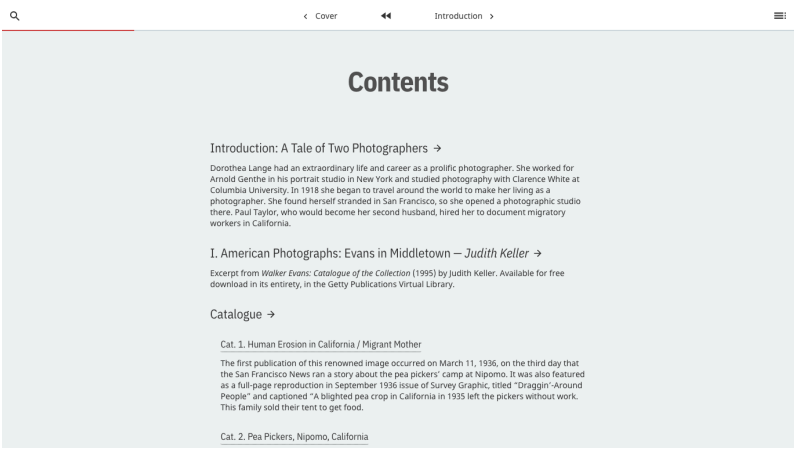
`type: page` (default)
The basic, default Quire page with title, page content, links and a list. A general publication page. Used for introductions, forewords, chapters, appendices and other pages. Also showing the progress bar at the top that indicates a reader's place in the publication.



`type: contents`
`class: list` (default)
The default contents page showing the title, subtitle and contributors for each main page and sub-section page. This page type automatically creates a table of contents for your entire publication, or for a section of your publication when used inside a sub-directory.

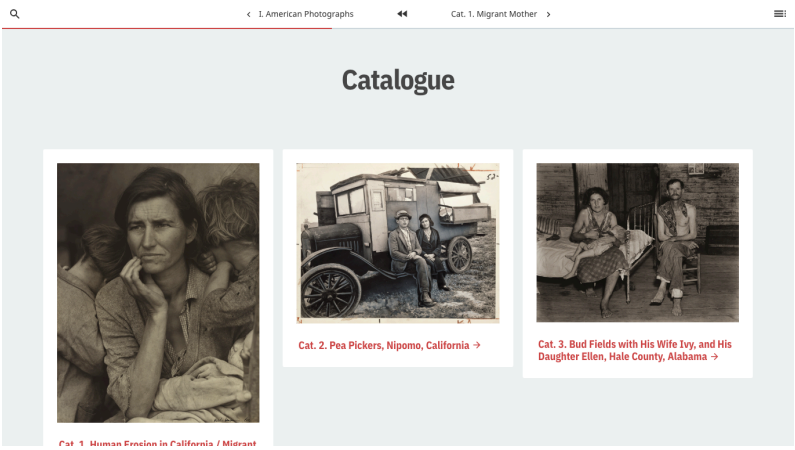


`type: contents`
`class: brief`
A minimal version of the contents page showing only the title for each main page. Sub-section pages have been optionally hidden using the `tocType: short` option in the config.yml file.



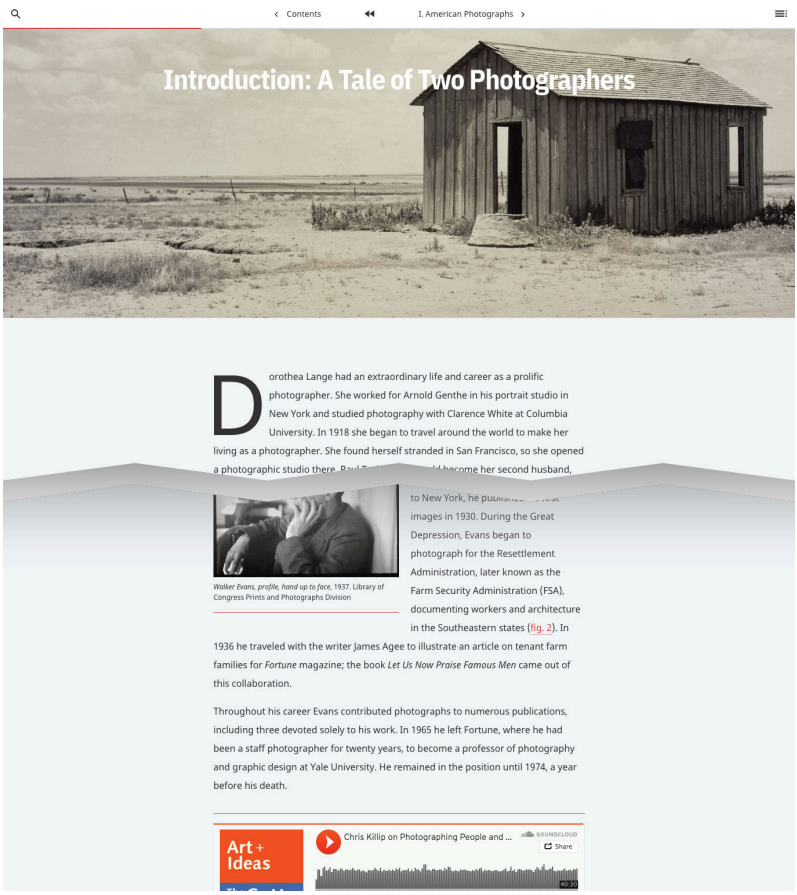
`type: contents`
`class: abstract`

A maximal version of the contents page showing the title, subtitle and contributors for each main page and sub-section page, along with a provided abstract or a generated snippet of the first part of the page's content.



`type: contents`
`class: grid`

A visual grid version of the contents. Displays an image if one is specified in the page YAML or if the page is an object entry page. Contents page types can also be used within sections to display the contents of that section, in this case, the Catalogue section.



`type: splash`

A splash page to open a section or to set off a particular page. Customizable banner image, drop cap lettering, full-color background. Also showing floating images and a Soundcloud embed.

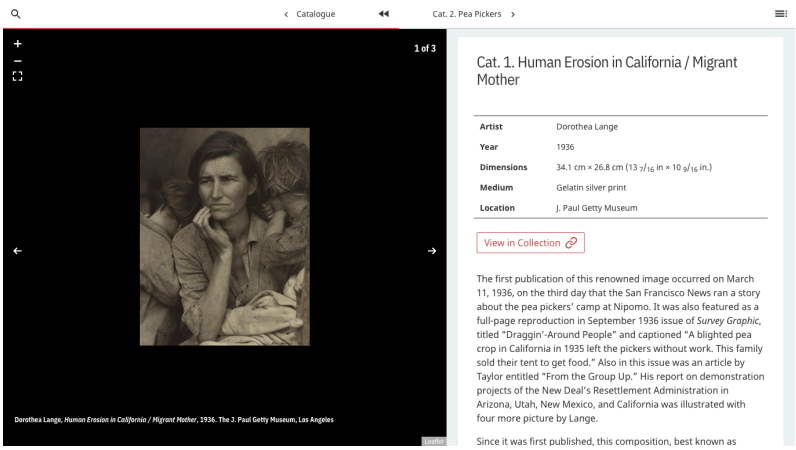
`type: essay`

An essay page showing a page label, contributor, abstract, figure group, links, and footnotes. The essay is a standalone, self-contained article in a periodical or collected volume. This is also reflected in the metadata embedded in the page, which will include more page-specific

information than a typical publication page, whose metadata will instead point to the publication as a whole.

Cat. 2. Pea Pickers, Nipomo, California

`type: entry`
`class: landscape` (default)
An entry page with entry text below and a nearly full-frame image viewer that supports zooming images, maps and videos.



`type: entry`
`class: side-by-side`
An entry page with scrolling entry text to the right side and a fixed-position image viewer to the left. All entry pages can be made to be in the side-by-side configuration by including `entryPageSideBySideLayout: true` in the config.yml file of the project.

Along with `type`, Quire pages can also have a `class`. These can be used to facilitate custom styling, but, as illustrated above, there are also a number of pre-defined classes that can be applied specifically to the `contents` and `entry` page types to give further control over the layouts of those pages.

```
type: contents
class: list (default) | brief | abstract | grid
```

```
type: entry
class: landscape (default) | side-by-side
```

Organize Pages in the Right Order

```
weight:
```

In the following example publication outline, we've listed the files and directories as we would like them to appear in the publication's table of contents.

```
cover.md
contents.md
part-one
  section-overview.md
  chapter-01.md
  chapter-02.md
part-two
  section-overview.md
  chapter-03.md
```

When looking in the actual `content` directory on your computer or in your text editor, however, they will almost certainly not appear in this order. More likely, they'll appear alphabetically or by date modified, which is also how Quire will order them when building and previewing your publication. You can adjust this by assigning a numerical `weight` to each page in its page YAML.

The page `weight` is a number and will determine the order in which the page appears in the publication. For example, the `contents.md` file in the example above, the second page in our book, would be `weight: 2`.

Numbering should be unique, and use sequential whole numbers, but it can skip numbers. So, if there's no page with `weight: 3`, Quire will

proceed to look for the next number. Intentionally skipping numbers in your sequence can be useful to leave room for adding content later. For example, your frontmatter might start at "0", your first section might be "100", second section "200" and so on. This makes it much easier to add a page to an early part of your publication, without renumbering every subsequent page.

> **TIP!**
>
> **Add `class: page-one` to the page/chapter where you want page 1 to start for the PDF/Print output. This is often an Introduction or first essay rather than the cover, table of contents, or other frontmatter.**

Create Section Landing Pages

A Quire publication can have sub-sections, created by nesting a group of one or more pages inside a sub-directory within the main `content` directory. It is recommended (though not required) to designate one of the pages in each sub-directory section to be the section landing page. To do so, add `slug: .` to the page YAML block. The `slug` attribute overrides the default name to be used in the URL for the page, and the period `.` refers it back to the sub-directory name. So, if in your site `mypublication.com` you have sub-directory called `part-one` and in that a landing page called `landing-page.md`, instead of the URL being `mypublication.com/part-one/landing-page/`, it would be `mypublication.com/part-one/`. Here's the YAML:

```
title: Part One
type: contents
class: grid
slug: .
```

The `title` of your defined landing page is what will be used in the header of that page, the *Table of Contents*, and the menu of your site.

However, the filename of the sub-directory itself is also used in your publication; for the online navigation bar, and in the running page footers of the PDF version. In these two places, Quire takes the sub-directory filename and humanizes it, which means to change hyphens into spaces and capitalize with title case. So, the sub-directory `part-one` becomes "Part One", or `sculpture-of-the-renaissance` becomes "Sculpture of the Renaissance."

Hide/Show Pages

By default, every page you create will be included in all formats of your publication (online, PDF/print, and e-book). Every page will also automatically be listed in the publication's menu and contents pages. However, this can be overridden by setting any of the following Page YAML attributes to `false`.

```
toc:
menu:
online:
pdf:
epub:
```

This allows you to do things like including an About page in your online edition, but a more traditional Copyright page in print. Or to substitute a simple splash page as a section break in the print, for the more elaborate contents grid you might use online.

TIP!

Note that when setting `online: false`, the page will not be included in the linear ordering of the book or in the menu,

table of contents, or search index, but it is still built. When deploying your site from the built files in the `/site/` directory, simply delete any unneeded ones. Read more about site deployment in the *Deploy Your Project* section of the documentation.

Page Content

Format publication content, including adding features and links

Format Text Content with Markdown

The main content of your page appears after the YAML block at the top (*Page Types & Structure*), and will be formatted in Markdown. Markdown is a very simple, plain text markup language that uses a few text rules to structure content for easy conversion into HTML. For example, a hash or pound sign at the beginning of a line makes a heading, and one set of asterisks wrapping around the text turns it *italic*.

The markdown file for this page starts like this:

```
---
title: Page Content
type: essay
weight: 206
---

## Format Text Content with Markdown
```

```
The main content of your page appears after the YAML block at
the top ([*Page Types & Structure*](/documentation/pages/)), and
formatted in Markdown. Markdown is a very simple, plain text
markup language that uses a few text rules to structure content
for easy conversion into HTML. For example, a hash or pound sign
at the beginning of a line makes a heading, and asterisks
wrapping text turns it *italic*.
```

You can read all about Markdown syntax and how it is used in Quire in the *YAML & Markdown* chapter of this guide.

Use Shortcodes to Add Features

Quire adds a number of specialty shortcodes which extend the functionality and possibilities of plain Markdown. While Hugo has a number of built-in shortcodes, which can also work in Quire, Quire-specific shortcodes always start with a `q`.

Shortcodes are always formatted with a combination of curly brackets and angle brackets with the name of the shortcode inside (`{{< q-shortcode >}}`) and often with some additional information in quotes. The example below inserts a figure in your document, matching a corresponding `id` with figure information stored in the publication's `figures.yml` file.

```
{{< q-figure id="3.1" >}}
```

While most Quire shortcodes work like `q-figure` as a single instance, the `q-class` shortcode acts as wrapper around other text and so it appears as a paired opening and closing shortcode. The closing code has a slash `/` preceding the shortcode name, much like you'd find in HTML markup. This example adds the class "alert" to the the phrase "Text goes here", which could be used to facilitate custom styling.

```
{{< q-class "alert" >}}
Text goes here
{{< /q-class >}}
```

TIP!

Quire includes one pre-defined class called "backmatter". This is typically used to wrap bibliographies, appendices, and other related content at the end of an article or page, and will style them to match the default footnote styling.

```
{{< q-class "backmatter" >}} ...
{{< /q-class >}}
```

The following shortcodes are currently available in Quire. You'll find more about them in their respective sections of the guide, as well as in the shortcodes api reference.

- `q-class` : As demonstrated above, wrapping text in this shortcode will allow you to apply a class name to that block of text, which can then be used to apply custom styles or interactions as needed.
- `q-bibliography` : Generates a bibliography from the entries in the project's `bibliography.yml` file.
- `q-cite` : Adds a linked Author Date citation reference to the text, and a hover pop-up with the full citation text. It also adds the citation to a map of cited works, which can then be output as a page-level bibliography on essay and entry type pages.
- `q-contributor` : Can be used to create a page of contributor biographies, a section of bios for a single page, a simple list of contributors, a byline for a particular page, or other similar outputs.

- `q-figure` : Inserts a formatted figure image (including audio and video) and caption using data from the project's `figures.yml` file, or from values supplied directly in the shortcode.
- `q-figure-group` : Like `q-figure` , but with handling for multiple images at once.

Apply Different Types of Links

As seen in the example above, a link is created by combining the text of the link in brackets with the url of the link in parentheses: `[Link text](Link URL)` There are several types of linking that can be applied to text on your page. Stylization such as bolding, italics, underlining, and more can also be applied to linked text.

External Links

External links can be included through the following Markdown formatting:

```
[Link text](http://www.linkaddress.com)
[Getty Museum](https://www.getty.edu/museum/)
```

These are set by default to open in new pages, but you can change that by setting `hrefTargetBlank` to `true` in the config.yml file.

Internal Links Between Pages

Internal links between pages in your Quire publication can be included through the following Markdown formatting using the file name of the page and the directory name of the section it is in.

```
[Link text](/name-of-section-if-any/nameofpage/)
[Pea Pickers](/catalogue/2/)
More info in our [about](/about/) page.
```

Internal Links to Specific Elements on Pages

There are several types of linking between features, text, or objects on a single page that can be included through the following Markdown formatting:

Links to Figures

This linking can be applied to a piece of text that when clicked upon will take a user to the location of the corresponding figure on the page. Figure IDs can be found on the `figures.yml` page as explained in the *Figure Images* chapter of this guide. They are proceeded by the # symbol when used as a link address.

```
[number or name of figure](#figureid)
[fig. 1](#1.1)
```

Links to Other Page Elements

An ID and the # symbol is also used for other kinds of elements on the same page. The IDs for these elements can be found using the following method:

- Use the Inspect Element tool when right clicking a page or specific element. For Safari users, refer to this guide to enable this feature.
- In the page's code, certain elements will include a piece of code, `id="idnamehere"` that designates the ID of that element. If the name of the element has a space that will be represented with a dash `-`.

- For example, the ID of a heading will often be the name of that heading.

```
[referencetolink](#element-id)
See [heading 1](#heading-1).
```

Links to Elements on a Separate Page

Following the formula for internal links between pages, you can also specify an element on a separate page as a link destination by adding the # symbol and the element's ID on to the end of a page link.

```
```  md
[referencetolink](/nameofpage/#idname)
See the introduction [notes](/introduction/#notes)
```
```

WARNING!

Blackfriday, Quire's built in Markdown processor, will incorrectly create link when there is some text in brackets followed immediately by more text in parentheses even if there is a space between them. To avoid the linking, you can use a \ (backslash) escape character before the first parentheses, such as: `[not a link] \(1926)` The \ will not display in the final rendered text.

Linked Footnotes

When creating footnotes with Markdown, links are automatically created between the footnote number in the text and the note itself at the bottom of the page. To link to a note from other locations, you can use its automatically generated ID, which always follows the format `fn:#` where # is the number of the footnote.

```
[referencetolink](#fn:#)
Also in regards to [note 21](#fn:21)
See [note 3, chapter 2](/chapter-2/#fn:3)
```

Citations

When the citation shortcode `{{< q-cite "author date" "page #" >}}` is used in a body of text and corresponds to the short and full bibliographic information provided in the references.yml file, an in-page bibliography will be generated and linked to. This linking is completed automatically.

When the shortcode is used in the page, the text will appear linked and when clicked upon will take a user to its corresponding bibliography entry on the same page. However, this cannot be done in reverse as the bibliography at the bottom of the page contains no links.

For more information see the Citations & Bibliography section of this guide.

Figure Images

Incorporate multiple images, videos, and other multimedia

Quire books are visual and the framework is built to support the use of images for scholarly purposes. On this page, we explain where images are placed in the project and how you can manage them. We recommend using the `figures.yml` file to manage all the information about your images, and then inserting them into your Markdown documents where they are needed with the `q-figure` shortcode. As a reminder, a shortcode is a simple snippet of code inserted in a Markdown file that pulls in information from other files in your project.

Include Figure Image Files in Your Publication

Figure image files should be placed in the `static/img/` directory. It is defined in your project's `config.yml` file with the parameter `imageDir: "/img/"` and the directory can be changed if needed.

[Note] You can organize figures into sub-directories within the `img` folder, but you will need to include those directories along with the filename when defining the `src` attribute for the figure, as noted below.

Quire does not require a specific image file format or size, but we have some recommended best practices:

- Use JPEG, PNG, or GIF.
- If your project is web-only, 800 pixels is fine for most images, whereas 1,800 pixels on the longest side will provide both a decent web experience and work for printing in the PDF without being too large a file size.
- Watch out for file sizes, especially on animated gifs which can get to be multiple megabytes quite quickly. Use Image Optimization software when possible, and consider the total number of images on a given page when choosing sizes.

To include deep-zooming images in your Quire project please see the *Zooming Images with IIIF* section of the documentation.

Create a figures.yml File for Figure Image Metadata

For most publications, or, at least, those with more than just a handful of images, figures and all their associated attributes can be listed in the `figures.yml` file, which should be placed in your `data` folder. This figure image metadata can then be called from wherever you need it in your project with a shortcode. See the API-DOCs section for complete details on possible figure attributes, but below there is a very simple example with `id` and `src` (required attributes) and `alt` (recommended attribute).

```
- id: "1.1"
  src: "clyfford-still_untitled96.jpg"
  alt: "detail of painting showing jagged brushstrokes in browns
- id: "1.2"
  src: "portrait-of-still.jpg"
  alt: "photograph of a frowning older man in brown jacket and f
```

Also available are the attributes `caption`, `credit`, `media_id`, `media_type`, `aspect_ratio`, and `label_text`.

> **WARNING!**
>
> **You can organize your images in the If your figures are organized in sub-directories within your `static/img/` directory. They should appear as part of the file path under `src`, otherwise, only the filename is needed.**

Insert Figure Images with q-figure Shortcode

Assuming each YAML figure entry in the `figures.yml` file includes a unique `id` (with a value in quotes: "1.1" not 1.1), you can insert a figure in your publication with only the `id` attribute in the shortcode, and all of the other attributes defined in the YAML for that figure will be automatically included.

Figure shortcodes should be inserted on their own line of your Markdown file, not within the text of a paragraph. A basic use of the `q-figure` shortcode would look like this:

```
{{< q-figure id="1.2" >}}
```

If you include an attribute in the shortcode that is also in the `figures.yml` file, the `figures.yml` version is overridden. This can be useful when, for example, a figure is used in multiple locations and you want different captions.

```
{{< q-figure id="1.2" caption="" >}}
```

TIP!

Leaving an attribute blank, as in the caption example above, can also be used to display no caption at all, even if one is present in `figures.yml`.

Attributes may be called within the shortcode in any order. `{{< q-figure id="1.2" caption="" >}}` is the same as `{{< q-figure caption="" id="1.2" >}}`.

Always use the figure shortcodes on their own lines in your Markdown documents, in between paragraphs. Never within a paragraph. Traditionally, figures will be placed directly after the paragraph in which they were first referred to.

Label Figure Images

By default, all figure images are labeled automatically, either at the start of the caption or just under the image itself in the case of a figure group with a single, group caption (see below). You can turn off this behavior in the `config.yml` file by switching the value `figureLabels: true` to `figureLabels: false`.

Figure labels are constructed with the `id` of the image and the `figureLabelsTextBefore` `figureLabelsTextAfter` values defined in your `config.yml` file. For example, if the `id` value is "12.3" and the `figureLabelsTextBefore` value is "Figure ", and `figureLabelsTextAfter` value is ". ", the resulting label would be "Figure 12.3".

To customize the label text on a figure-by-figure basis, use the `label_text` attribute in the YAML attributes for your figure. Any text there will override the automatically constructed version.

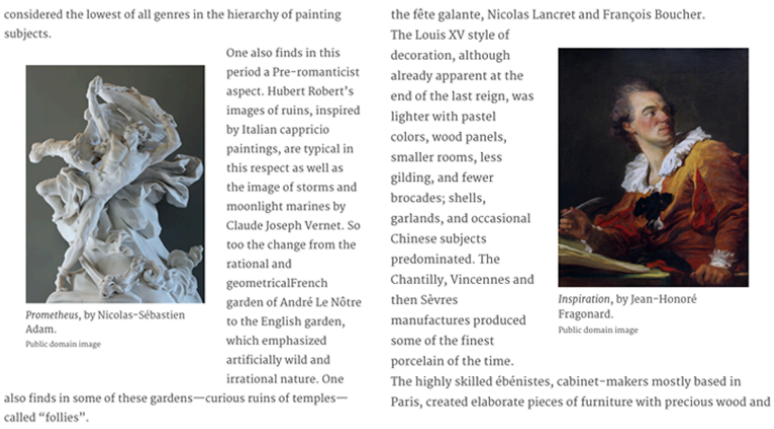
To remove a label from a specific figure or a group of figures, add `label="false"` to the shortcode. Or, in reverse, if you already have

`figureLabels: false` set in your `config.yml` file, use `label="true"` in the shortcode to show a label for that figure.

Style Figure Images

Depending on your theme, by default, figures will appear at about the width of the full-column of text. Modifier classes can be added to a shortcode to style the way the figures appear. Available classes are `is-pulled-left` and `is-pulled-right`. Classes are added just like other attributes in the shortcode.

```
{{< q-figure id="1.2" class="is-pulled-left" >}}
```



considered the lowest of all genres in the hierarchy of painting subjects.

Prometheus, by Nicolas-Sébastien Adam.
Public domain image

One also finds in this period a Pre-romanticist aspect. Hubert Robert's images of ruins, inspired by Italian cappricio paintings, are typical in this respect as well as the image of storms and moonlight marines by Claude Joseph Vernet. So too the change from the rational and geometricalFrench garden of André Le Nôtre to the English garden, which emphasized artificially wild and irrational nature. One also finds in some of these gardens—curious ruins of temples—called "follies".

the fête galante, Nicolas Lancret and François Boucher. The Louis XV style of decoration, although already apparent at the end of the last reign, was lighter with pastel colors, wood panels, smaller rooms, less gilding, and fewer brocades; shells, garlands, and occasional Chinese subjects predominated. The Chantilly, Vincennes and then Sèvres manufactures produced some of the finest porcelain of the time.

Inspiration, by Jean-Honoré Fragonard.
Public domain image

The highly skilled ébénistes, cabinet-makers mostly based in Paris, created elaborate pieces of furniture with precious wood and

---

TIP!

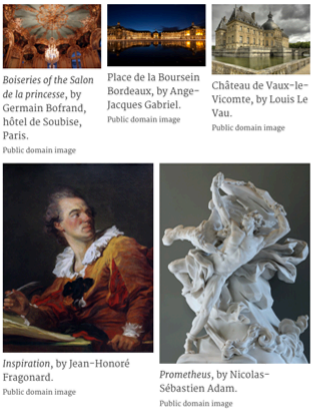
Some themes may offer additional options, and styles may be edited and new styles added in any theme with CSS.

Create and Style Figure Groups with q-figure-group Shortcode

If your project uses a `figures.yml` file, you can also create a group of figures by using the `q-figure-group` shortcode and simply including multiple, comma-separated values in the `id` field.

```
{{< q-figure-group id="1.1, 1.2" >}}
```

In the above example, each figure's caption will be included in the grouping. Alternatively, if you add a `caption` attribute directly in the shortcode, it will override those present in the `figures.yml` file and display with the group alone as a single, group caption.



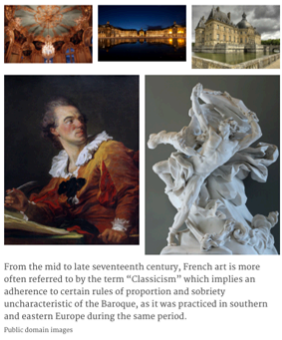
Just as with the single `q-figure` shortcode, classes can be added to groups to style them. For example, to create a small group of images running along one side of your text.

```
{{< q-figure-group class="is-pulled-left" id="1.1, 1.2" >}}
```

In addition to all the attributes available to the `q-figure` shortcode, the `q-figure-group` extension also supports the `grid` attribute to specify a preferred grid width. In the below example, a `grid="2"` is specified and so the gallery grid will be 2 images wide at your publication layout's full-size. Alternately, if you specified `grid="4"` the grid would be 4 images wide, making each image relatively smaller.

```
{{< q-figure-group grid="2" id="1.1, 1.2, 1.3, 1.4" >}}
```

`grid="2"`



`grid="3"`

`grid="4"`

WARNING!

Note that this is only a preferred grid width. With Quire's responsively designed templates, the specific width of images is variable and their position relative to one another may also change depending on a reader's device. For instance, on a large monitor, four images in a group may appear side-by-side in a row, whereas on a phone, they would most likely be in a 2 x 2 grid, or stack one on top of another. This responsiveness also means that group captions that use language like "From left to right" or "Clockwise from upper left," will only be correct some of the time. To avoid this issue and ensure a clear reading experience across all devices and publication formats we recommend labeling figures individually.

Add Video Figures

Videos can be embedded in your publication the same way as other figure images, using either of the two figure shortcodes. The difference is in the `figures.yml` file where you'll need to include a `media_id` and a `media_type` attribute for any video, along with an optional `aspect_ratio` attribute.

Quire supports video embeds from either YouTube (`media_type: youtube`) or Vimeo (`media_type: vimeo`). The `media_id`s can be found in the URLs of the videos you wish to embed. For example, in https://www.youtube.com/watch?v=VYqDpNmnu8I or https://youtu.be/VYqDpNmnu8I, the `media_id` would be `VYqDpNmnu8I` ; and in https://vimeo.com/221426899 it is `221426899` .

```
- id: 1.5
  src: videostill.jpg
```

```
media_id: VYqDpNmnu8I
media_type: youtube
```

TIP!

The `src` image provided in this example is a frame from the video and will be used in place of the video in the PDF and EPUB versions of your publication. In Quire this is referred to as a fallback. Along with the fallback image, Quire will also automatically append a link to the video following the caption.

Like the image labels this is controlled in the project's `config.yml` file with `videoFigureFallbackText: true`, `videoFigureFallbackTextBefore: "Watch the video at "` and `videoFigureFallbackTextAfter: "."`.

TIP!

Note that on YouTube, videos can be filed as "Unlisted" and this will let you embed the video, but will not include the video on your channel page, or in YouTube's general search engine.

Add Basic Figures

If you are not using a `figures.yml` file, figures—including still images and animated gifs but not video—can be inserted in any Markdown document in your publication with the `q-figure` shortcode, where `src` is the name of your file as it appears in the `static/img/` directory of your project.

```
{{< q-figure src="fig01.jpg" >}}
```

Additionally, you can add `caption`, `credit`, `class`, and `id` attributes in this manner.

Unless the figure is purely decorative, it should always also include an alternate textual description (`alt`) for the use of screen readers and other assistive technologies. We recommend using alternate textual description for accessibility purposes. For more information check our Accessibility Principles

```
{{< q-figure src="fig01.jpg" alt="detail of painting showing dia
```

Zooming Images with IIIF

Adding high-resolution zooming images to your project

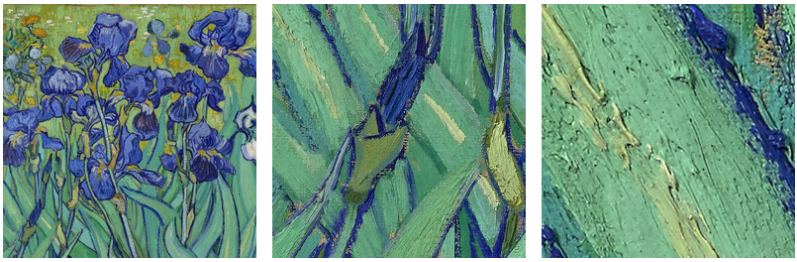
To add high-resolution zooming images to your project, Quire uses the International Image Interoperability Framework (IIIF). Within your Quire project, you can either point to existing IIIF assets or create your own. IIIF zooming images can be displayed within your Markdown pages using the `q-figure-zoom` shortcode or the built-in image viewer on entry pages. (Learn more about entry pages in the *Collection Catalogues* section of this documentation.)

What is IIIF?

IIIF (referred to when speaking as "Triple I F") stands for the International Image Interoperability Framework. It is a shared, open set of standards for storing images and image data that allows for seamless and regularized sharing and display of those images across different uses within an institution or across different institutions. While IIIF images offer a wide variety of features and applications, Quire uses them specifically for deep zooming.

A sample high-resolution IIIF image (*Irises*, Vincent Van Gogh, 1889. J. Paul Getty Museum.) with six levels of zoom, created from a source image that was 8,448px on the longest side using the `quire process --iiif` command. The image files for this are in the `static/img/iiif/processed` directory of this Quire publication.

You can either point Quire to an existing IIIF repository of images or add high-resolution images into a Quire project directly. When adding your own images, Quire processes them into hundreds of individual image tiles. Each tile comprises a portion of the image at a particular zoom level to then be displayed in our existing zooming image viewer. Because the images have been tiled, users of your Quire project only need their browsers to load the tiles of the image portion and zoom level they're looking at at the moment. This means much faster page loads and higher-resolution images.



Three IIIF image tiles from the same painting, at three different levels of zoom. Each zoom level can be comprised of hundreds or even thousands of individual tiles.

TIP!

For a deep dive into IIIF and its implementation in museums read *Getty Common Image Service: Research & Design Report*.

Use Existing IIIF Images

To include existing IIIF images in your project, you will need the URL for the `info.json` file for that image, as well as a static fallback version of the image (typically a JPG) for use in Quire's PDF and e-book outputs.

The `info.json` file is a JSON-formatted collection of useful information a file. It is a required component of all IIIF images and includes basic object data about the full image and the availability of image tiles at different zoom levels.

If you are working with your own institution's IIIF repository, your digital department will be able to help you identify the `info.json` URLs you need.

If you are working with other, open IIIF repositories, more often than not, you will find a URL to the IIIF `manifest.json` file rather than an `info.json` file. For instance, the Art Institute of Chicago has the IIIF Manifest URL available in plain text on every page of its collection where IIIF is available. The Getty Museum instead includes a IIIF logo on the

artwork page, from which you can extract the `manifest.json` URL which appears at the end of the URL when you click.

The `manifest.json` file is a more fullsome IIIF document that includes much more information about the image and the resources available. Within that `manifest.json` file however, you can look under `sequences[0].canvases[0].images[0].resource.service.@id`. The URL in the `@id` field, when appended with `/info.json`, will be what you need to add the image to your Quire project.

In this example `manifest.json` file, the URL in the last `@id` field is what we're looking for.


```
{
  "@context": "http://iiif.io/api/presentation/2/context.json",
  "@id": "https://data.getty.edu/museum/api/iiif/826/manifest.js
  "@type": "sc:Manifest",
  "label": "Irises (1889), Vincent van Gogh (Dutch, 1853 - 1890)
  "sequences": [
    {
      "@id": "https://data.getty.edu/museum/api/iiif/826/sequenc
      "@type": "sc:Sequence",
      "canvases": [
        {
          "@id": "https://data.getty.edu/museum/api/iiif/826/can
          "@type": "sc:Canvas",
          "width": 9073,
          "height": 7134,
          "images": [
            {
              "@id": "https://data.getty.edu/museum/api/iiif/826
              "@type": "oa:Annotation",
```

```
            "motivation": "sc:painting",
            "resource": {
              "@id": "https://data.getty.edu/museum/api/iiif/6
              "@type": "dctypes:Image",
              "format": "image/jpeg",
              "service": {
                "@context": "http://iiif.io/api/image/2/contex
                "@id": "https://data.getty.edu/museum/api/iiif
                "profile": "http://iiif.io/api/image/2/profile
              },
              "width": 9073,
              "height": 7134
            }
          }
        ]
      }
    ]
  }
}
```

Test it by pasting it into a browser, adding `/info.json`, and hitting enter. If correct, you'll get a complete JSON document that includes:

```
"@context":"http://iiif.io/api/image/2/context.json"
"@type":"iiif:Image"
```

Once you've identified the URL to the proper `info.json` file, you can jump to the section below on displaying IIIF image in your project.

> **WARNING!**
>
> **When using external resources, IIIF or otherwise, be aware that changes made to those resources by their hosts can affect the way they display in your project. Always link to reliable, permanent sources, and have a regular maintenance schedule to check for unforeseen changes that may occur after publication.**

Create Your Own IIIF Images

You can create your own deep-zooming IIIF images directly within Quire. But there are a couple things you should know from the outset:

1. **Processing IIIF image tiles is a labor intensive process.** It is highly dependent on the processing power of the computer you're working on. We've found that processing a single 130MB image on an average to good machine can take upwards of 20 minutes or more. Three 6–7MB images may take only two or three minutes. Dedicate time to IIIF processing in batches when you're not running other software on your computer.

2. **You may need to setup an additional hosting service for IIIF image files, outside your main project files.** Though small in individual file size, the sheer quantity of files (hundreds and sometimes thousands) that are associated with deep-zooming images can cause preview and build issues within your Quire project. If you have more than around 12 IIIF images you'll want to host them outside your project.

1. Prepare High-Resolution Source Images

Because the IIIF image tiles are ultimately saved as JPEGs, **we recommend starting with uncompressed, full-quality JPEG (jpg) or JPEG 2000 (jp2/jpf/jpx) files for IIIF processing**. TIFs can also work, but are typically much larger file sizes and don't necessarily provide better end results. Quire can also process SVG and PNG files.

Use the table below to size your source images prior to IIIF processing. Images that deviate from the recommended size guidelines for source images can also end up with misalignments and glitches at some zoom levels. The longest pixel dimension should be divisible by the 256px tile size, and be as close as possible to the final tiled image size without being equal to that size.

TIP!

Save some time! Images larger than the recommended size for a zoom level will work but will be downsized to the final image size regardless and will cause the image processing to take longer due to the excess file size.

Desired Zoom Levels	Final Size of Tiled Image on Longest Side	Recommended Size of Source Image on Longest Side
4	2,048px	2,304px
5	4,096px	4,352px
6	8,192px	8,448px
7*	16,384px	16,640px
8*	32,768px	33,024px

** Currently, Quire is currently set to do a maximum of 6 levels of image tiling. This can be expanded but requires a code change to both the Quire CLI and the JavaScript of your project's theme.*

For images smaller than 2,304px on the longest side, we recommend including them as regular non-IIIIF images. Typically, a JPEG of around 1,800px on the longest side, and at 70–80% quality, will provide both a decent web experience and work for printing in the PDF, without being too large of a file size. Read more in the *Figure Images* chapter of the documentation.

2. Process Images Into IIIF Tiles

Once they've been properly sized following the guidelines above, high-resolution source images should be placed in the `static/img/iiif/` directory of your project. The folder can be created if it doesn't already exist in your project. It's best to start with only 2–3 images to test how your computer will handle the processing.

In your command-line shell, run the following command:

```
quire process --iiif
```

Quire will then start to process the high-resolution images into IIIF image tiles. It will output warnings if it encounters any files it can't process, as well as messaging for when each image is completed. Quire will process all images in the `static/img/iiif/` directory, including ones that have been previously processed. So, once a batch of high-resolution files are processed, it's best to remove them from the directory to prevent duplicate processing.

The processed IIIF files will be placed in a `static/img/iiif/processed` directory with folder names matching the original filenames of the high-resolution images. Each folder includes the hundreds or sometimes thousands of individual image tiles arranged in their own directories that make up the zoomable image. The directory will also include an

`info.json` file, which includes basic information about the full image and the availability of image tiles at different zoom levels.

You will need the path to the `info.json` file to include it in your `figures.yml` file when getting ready to display your IIIF images in your project. The paths are also based on the filename of the original high-resolution you processed. For example, if your high-resolution image was `00094701.jpg`, the path to the `info.json` would be `/img/iiif/processed/00094701/info.json`.

3. Host the IIIF Image Tiles

IIIF image tiles can be hosted statically (meaning you don't need a special server setup), just like the website edition of your Quire project. In fact, a modest number of IIIF images (around a dozen or so) can even be hosted within the project itself. In these cases, the IIIF files you've processed with the `quire process --iiif` command can be left where they are. It is also fine to move the processed IIIF image folders into different locations within your `static` directory. You will just need to update the paths to their `info.json` files accordingly when you're adding them to your `figures.yml` file, as explained in the next section.

With more than twelve or so IIIF images, you're going to need to host them elsewhere and then point to them from your project. This is because, though small in individual file size, the sheer quantity of files (hundreds and sometimes thousands) that are associated with deep-zooming images can cause issues when trying to run `quire preview` or use GitHub or a related service to host your project code.

TIP!

When quantifying IIIF images, it's hard to say how many IIIF images will be too many for a system to handle, because the number of individual files associated with a given IIIF image is highly variable.

For example:

4 levels of zoom ≈ 100 image tiles
5 levels of zoom ≈ 329 image tiles
6 levels of zoom ≈ 1,170 image tiles

For hosting the image tiles, if you have institutional support, your digital department should be able to provide a solution. If you're on your own, there are any number of options. You might check out Amazon S3, which is very performant and self-serve but requires some technical savvy to get set up, or an independent hosting service like Reclaim Hosting, which caters to the academic sector.

Display IIIF Images in Your Project

To display a IIIF image in your project, you need to point to a IIIF `info.json` file for the image. Either one you've identified from an external source, or one you've created yourself with `quire process --iiif`. If it is an image you processed yourself and it is still in the `processed/` directory, the path to the JSON file would be `/img/iiif/processed/FILENAME/info.json`.

Along with the path to the `info.json` file, you also need to include `media_type: "iiif"` and a path to a lower-res static fallback version of the image, hosted in your project's `static/img` directory (which is used for Quire's PDF and e-book outputs). All of this goes in your `figures.yml` file.

```
- id: "irises"
  src: "figures/irises.jpg"
  media_type: "iiif"
  iiif: "https://data.getty.edu/museum/api/iiif/671108/info.json
```

The image can then be added to your Markdown files using the `q-figure-zoom` shortcode, as shown below. This will display the static image on the page, and then when clicked, will open the figure viewer for the fully zoomable IIIF version. (See the page on *Figure Images* for more information on the `figures.yml` file and other figure shortcodes.)

```
{{< q-figure-zoom id="irises" >}}
```

IIIF Images can also be displayed in the built-in image viewer on entry pages. Do this by including the appropriate `id` in the object data in your `objects.yml` file.

```
object_list:
  - id: 1
    title: *Irises*
    artist: Vincent van Gogh
    year: 1889
    medium: Oil on canvas
    dimensions: 74.3 × 94.3 cm (29 1/4 × 37 1/8 in.)
    location: J. Paul Getty Museum
    link: https://www.getty.edu/art/collection/objects/826/vince
    figure:
      - id: "irises"
```

Include a Static Fallback Image

The static fallback image is required for displaying a version of the IIIF image outside of the zooming image viewer on the online version of your project, as well as for Quire's PDF and e-book outputs. This is the standard `src` attribute in your `figures.yml` listing.

```
src: "figures/irises.jpg"
```

Like with other non-zooming figure images, these static images for IIIF would typically be a JPEG of around 1,800px on the longest side and 70–80% quality. These specs provide both a decent web experience and work for printing in the PDF without being too large of a file size. Read more in the *Figure Images* chapter of the documentation.

Set the Zoom Level

Quire will, by default, show six levels of zoom. If the IIIF image you are pointing to has less than that, you can set the number of levels in `figures.yml` with the `zoom_max` property. This will ensure your users can't zoom further in than the image is sized for, which would result in blurred images.

```
- id: "irises"
  src: "figures/irises.jpg"
  media_type: "iiif"
  iiif: "/img/iiif/processed/07138601/info.json"
  zoom_max: 4
```

The number of zoom levels will be determined by the size of the source file, but you can also tell by looking a the processed image in the `static/img/iiif/processed/` directory. The number of directories there that start with "0,0," will be equal to the number of zoom levels. In the following example, image `07138601` has four levels of zoom.

```
static
  img
    iiif
      processed
        07138601
          0,0,256,256
          0,0,512,512
          0,0,1024,1024
          0,0,2048,1919
          0,256,256,256
          0,512,256,256
          0,512,512,512
          0,768,256,256
          ...
```

Currently, six levels of zoom is the maximum Quire is set to display. This can be expanded within your individual project, but requires a code change to both the Quire CLI and the JavaScript of your project's theme.

Maps

Tips and tricks for including basic, interactive maps

WARNING!

Basic mapping functionality was added into Quire early on but hasn't been used in production on any projects yet, and so remains very rough. Maps are unlikely to be a supported feature moving forward as they can be very demanding and beyond what we can support for the Quire community at large. In the future, we hope to demonstrate some working examples that can act as a guide if you wish to pursue maps yourself. In the meantime, you're welcome to make use of the map functionality as it exists but proceed through this section with caution.

Start with an entry in your `figures.yml` file. It must include `media_type: map`, latitude (`lat`) and longitude (`long`) points to define where you would like your map centered, a `src` for a static image of your map, and the path to a `geojson` file. When starting a new Quire project, you will find a sample geojson file in `/static/data/sample-geojson.json`.

```
- id: "map"
  lat: 48.8566
  long: 2.3522
  src: figures/map.png
  media_type: map
  geojson: /data/sample-geojson.json
  caption: "Great cities in France."
```

Geojson data is what powers the marked points, regions, and labels on your map. There are several tools for creating and editing it and sites to help you understand it. Quire uses Leaflet as a map viewer, so some of their documentation should be applicable here. The only thing we're demonstrating in the current, basic mapping functionality are marking points and adding a pop-up to them with some text.

Basic geojson:

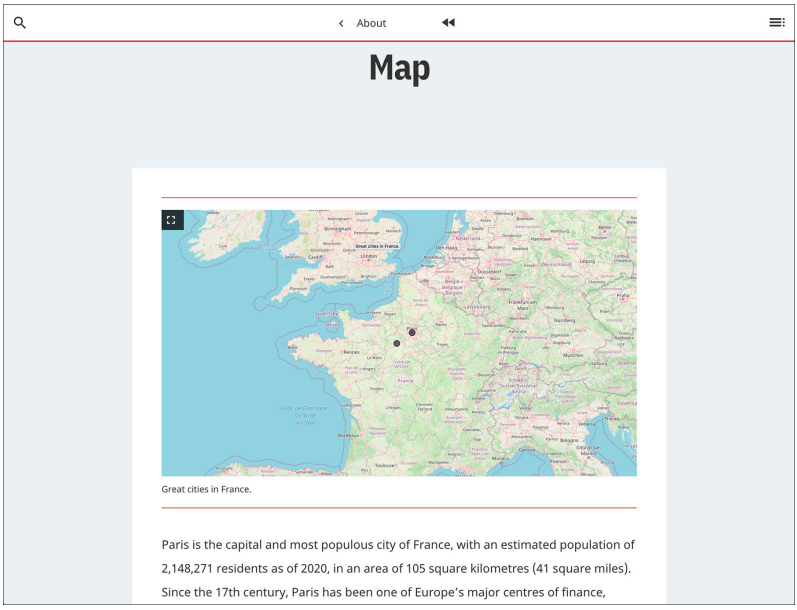
```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [2.3522, 48.8566]
      },
      "properties": {
        "name": "Paris",
        "description": "The city of lights"
```

```
      }
    }, {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [1.map-inlinemap-inline4695228, 48.448102
      },
      "properties": {
        "name": "Chartres",
        "description": "Southwest of the city of lights"
      }
    }
  ]
}
```

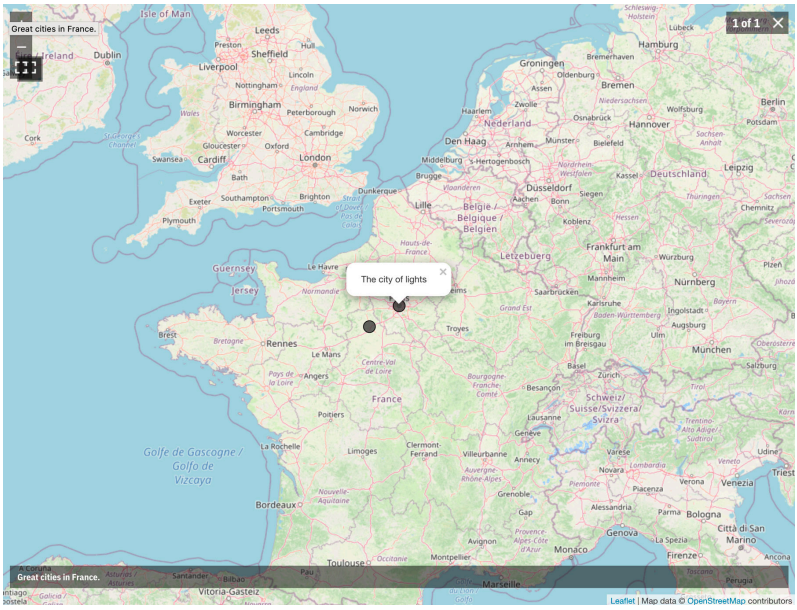
Once you have your entry in `figures.yml` and it's pointing to your geojson data, you can add a map to the page with the `q-figure-zoom` shortcode:

```
{{< q-figure-zoom id="map" >}}
```

The result is a figure image with the static map image and any provided caption, which can be clicked on to open the full, interactive map with zooming and your marked geojson points.



When you insert an interactive map with the `q-figure-zoom` shortcode, it will display a static image of the map when you're on the page.



When you click on a map inserted with the `q-figure-zoom` shortcode, it will load an interactive version centered on latititue and longitude points you specified, and displaying labels and points from your geojson data file.

You'll see there are some issues with the interface here (such as the duplicated full screen icon in the upper right), and the city labels we included in the geojson for Paris and Chartres are not included on the map points, nor as part of the pop-up label.

The map is fully zoomable and global. The open-source images come from OpenStreetMap. Just note that this is an external resource to your Quire project and is subject to change within your publication over time as OpenStreetMap updates its maps or changes its service.

For the advanced user, it is possible to further configure your map, and potentially point to different map image tiles in `/themes/default/source/js/map.js` .

Citations and Bibliographies

Cite sources with pop-ups and generate reference lists

In-text citations and bibliographies are all available in Quire. Designed to meet scholarly needs and multiple citation styles, they are easy to implement in your publications. While bibliographic references are formatted in YAML and stored in a YAML file (you can consult our YAML syntax fundamentals for more information), citation and bibliography shortcodes are used to integrate the references in your publication.

Capture Bibliographic Information in YAML

Bibliographic references for your publication can be listed in a `references.yml` file in the `data` directory (along with the `publication.yml`, `figures.yml` and `objects.yml` files).

Each entry in the `references.yml` file should include a `full` form of the reference, and then an `id` to reference it by.

```
entries:
  - id: "Faure 1909"
    full: "Faure, Élie. *Histoire de l'Art*. Vol. 1, *L'Art anti
  - id: "de Goncourt 1851"
    full: "de Goncourt, Edmond. *Journal des Goncourt: Mémoires
```

As in the example above, the `id` would typically be the short form of the reference in author-date format. However, you can also specify the short form as a separate line of the YAML.

```
entries:
  - id: "fre09"
    short: "Faure 1909"
    full: "Faure, Élie. *Histoire de l'Art*. Vol. 1, *L'Art anti
  - id: "degncrt51"
    short: "de Goncourt 1851"
    full: "de Goncourt, Edmond. *Journal des Goncourt: Mémoires
```

Sort Order

By default, when they are output in your publication Quire will sort the entries alphabetically by the text of the `full` reference. However, there may be some times when the sort does not output the expected results, in which case you can specify a specific `sort` value.

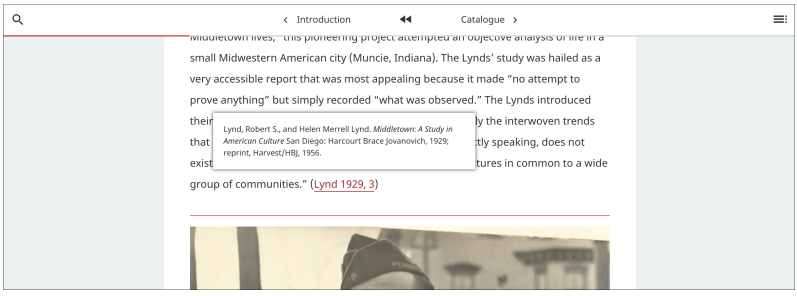
```
entries:
  - id: "fre09"
    short: "Faure 1909"
```

```
    full: "Faure, Élie. *Histoire de l'Art*. Vol. 1, *L'Art anti
    sort: "faure"
  - id: "degncrt51"
    short: "de Goncourt 1851"
    full: "de Goncourt, Edmond. *Journal des Goncourt: Mémoires
    sort: "goncourt"
```

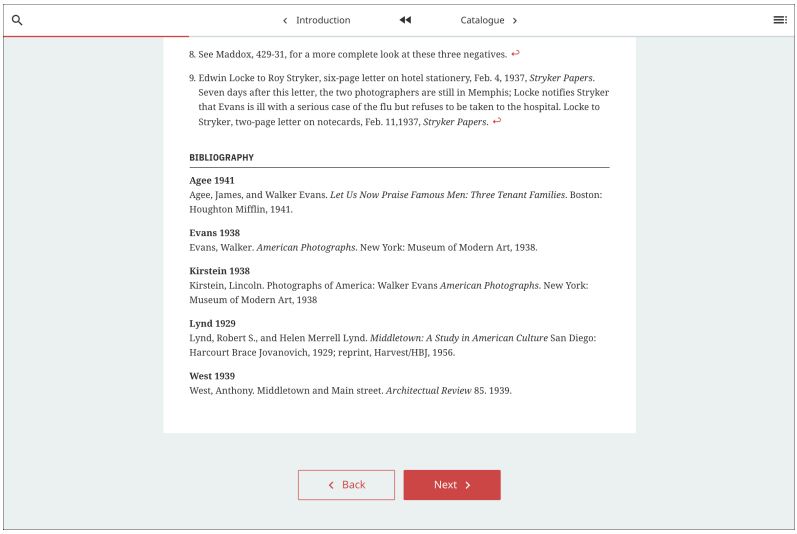
These references can then be called individually from within text using the `q-cite` shortcode, or in their entirety as a generated bibliography using the `q-bibliography` shortcode. Both of which are detailed below.

Add Inline Text Citations

The `q-cite` shortcode adds a linked Author Date citation reference to the text, and a hover pop-up with the full citation text. It also adds the citation to a list of all cited works on that page, which is output as a page-level bibliography, as explained below.



By using the `q-cite` shortcode, you can add citations that appear when hovering over the linked text. Clicking the link brings you to a bibliography at the bottom of the page.



Any citations added to a page with `q-cite` are automatically added to a bibliography list at the bottom of the page.

The first positional parameter of the `q-cite` shortcode is a short form citation that should match one in `references.yml`. The second, *optional* parameter is a page reference. The following sample would output as: Faure 1909, 54.

```
{{< q-cite "Faure 1909" "54" >}}
```

A third optional parameter allows you to customize the text to appear in the link if not the short form of the citation. The following sample would appear simply as: 1909, 54.

```
{{< q-cite "Faure 1909" "54" "1909" >}}
```

In using this third parameter, you still need to have the second parameter even if it's empty. The following sample would appear simply as: 1909.

```
{{< q-cite "Faure 1909" "" "1909" >}}
```

The text element between the author date reference and the page can be changed with the `citationPageLocationDivider` property in `config.yml`. The humanities tend to favor comma separation (which is the default in Quire), whereas the sciences typically favor a colon.

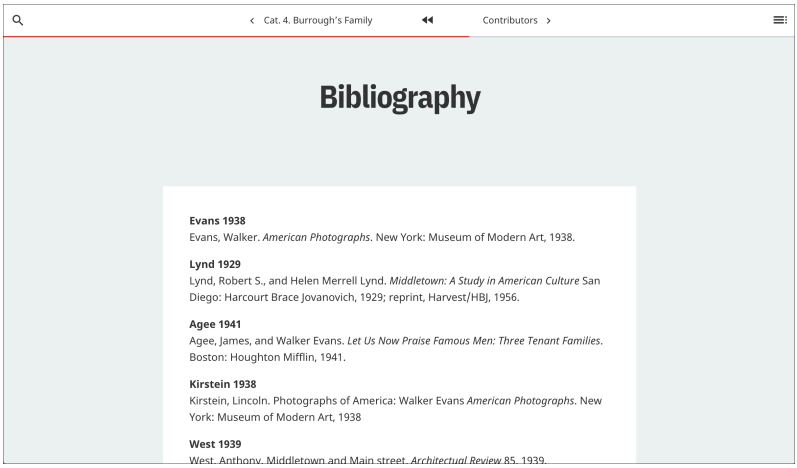
The `q-cite` shortcode can be used anywhere in your Markdown text, including within footnotes.

Display a Bibliography

Pages in your publication will automatically include a page-level bibliography listing all works that were cited on that page using the `q-cite` shortcode. A heading can be customized to go above the page bibliography with the `biblioHeading` parameter in your `config.yml` file. The default is “Bibliography”.

Additionally, to create a complete bibliography for your entire publication, from all the entries in the project’s `references.yml` file, you can use the `q-bibliography` shortcode. The resulting bibliography will be output in the order in which it appears in the references file.

```
{{< q-bibliography >}}
```



A bibliography of all works in your project's references.yml file can be added to any page with the `q-bibliography` shortcode.

This shortcode accepts an optional `sort` value, which will sort the list by whatever key from the entries is given. Often `"short"`, `"full"`, or `"sort"`, though a custom key could be added. Without a `sort` value given, the bibliography will be output in the order in which it appears in the references file.

```
{{< q-bibliography sort="short" >}}
```

You may find in some cases that the system's default sort method is suboptimal. In particular, the sort is case sensitive and will sort uppercase, before lower. So a reference for "e.e. cummings" would be listed after those for "Emily Dickinson". In these cases a custom key like `"sort"` could be added to all entries in the `references.yml` file for fine-grained control.

```
entries:
  - short: "cummings 1914"
    sort_as: "cummings-e-e"
  - short: "Dickinson 1932"
    sort_as: "dickinson-emily"
```

WARNING!

If adding a custom sort key, it would need to be added to *all* entries, not just the one that need to be sorted differently than the default.

Display the Short Reference in Bibliographies

Bibliographies displayed automatically at the bottom of pages, and those generated with the `q-bibliography` shortcode, can be a list of the full version of the reference, or can include the short version as well. This is controlled globally (all bibliographies in the project have to be the same format) in the `config.yml` file with the `displayBiblioShort` property, can be set to `"true"` or `"false"`.

Collection Catalogues

Learn how to publish a collection catalogue with Quire

Along with monographs, edited volumes, and serial publications, Quire is also designed with the publication of museum collection catalogues in mind and has a specific page `type` for them (See all page types in the *Defining Page Types* section of the *Pages and Plain Text* page of this guide). Collection catalogues typically feature a page for each object, featuring images of the object, information about it, and an essay or entry text. To publish a catalogue with Quire, you'll capture each object data, create the object pages, and then, optionally, display a list of the objects included in your publication. Essays in object pages work in the same way as any other pages, and you can visit our *Markdown fundamentals* page for reference.

Capture Object Data

Much like `figures.yml` or `references.yml`, all catalogue object metadata should be captured in a single `objects.yml` file in the `data` directory of your project and then labeled as needed in different pages of your publication. Here is a brief sample:

```
object_display_order:
  - artist
  - year
  - dimensions
  - medium
  - location
object_list:
  - id: 2
    title: Impression, *Sunrise*
    artist: Claude Monet
    year: 1872
    medium: Oil on canvas
    dimensions: 48 cm × 63 cm (18.9 in × 24.8 in)
    location: Musée Marmottan Monet, Paris
    link:
    figure:
      - id: "cat2"
  - id: 3
    title: Reading (portrait of Edma Morisot)
    artist: Berthe Morisot
    year: 1873
    medium: Oil on fabric
    dimensions: 74.2 x 100.3 x 12 cm (29 3/16 x 39 1/2 x 4 11/16
    location: Cleveland Museum of Art
    link: http://www.clevelandart.org/art/1950.89
    download: true
    figure:
      - id: "cat3"
```

```
    - id: "cat3a"
    - id: "cat3b"
```

There are two sections in the `objects.yml` file: `object_list` and `object_display_order`:

- The `object_list` is a list of the objects and their individual metadata attributes. With the exception of a few reserved terms, as noted in the table below, any attributes can be included here. These attributes and the associated values will ultimately display on the entry pages for each object.
- You control the specifics of which attributes to display and in what order by listing them under `object_display_order`. Following the sample above, the attributes included on the pages would be: `artist`, `year`, `dimensions`, `medium`, and `location`.

  Any images of the object are also included here, under the `figure` attribute. This is a list of one or more images. It is recommended that this list contains only `id` values corresponding with `id`s in your project's `figures.yml` file. However, if you prefer, you can instead include a `src` attribute with the filename as it appears in your project's image directory.

Here are the only defined object attributes, you can include any others you like:

Attribute	Description
`id`	Required. Used to reference objects from entry pages. Should be numbers and lowercase letters only, with no spaces or special characters (`001`, `fig-01a`, etc).
`figure`	A list of one or more images of the object. It is recommended that this list be only of `id` values

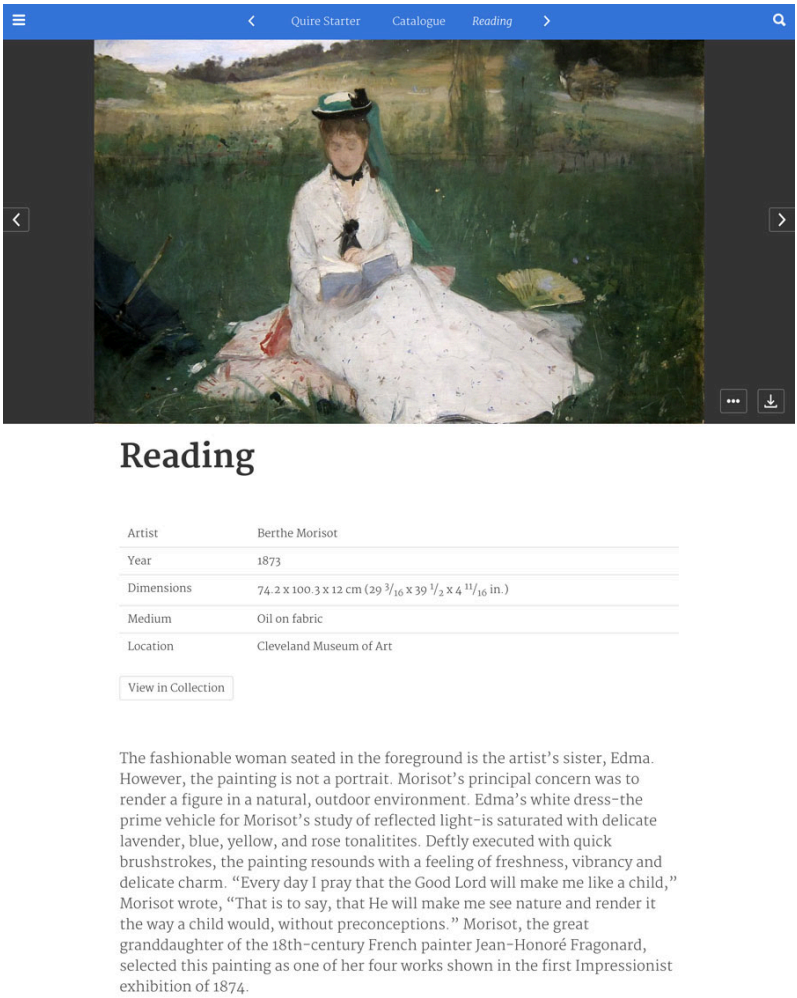
Attribute	Description
	corresponding with `id`s in your project's `figures.yml` file.
`link`	A URL link to a page with more/current information on the object. Usually the object in the museum's online collection pages.
`date_start`, `date_end`	Reserved for future use in Quire.
`dimension_width`, `dimension_height`, `dimension_depth`	Reserved for future use in Quire.

Create Object Pages

Like all other pages in your publication, object pages are generated from the Markdown files in your `content` directory. To create an object entry page, give the page a `type: entry` in the page YAML block, and list one or more objects by `id` corresponding to those in your `objects.yml` file.

```
type: entry
object:
  - id: 1
```

The page will feature any images associated with the object, followed by a table of object information and finally an essay/entry text included in the page Markdown file.



screenshot of catalogue entry page as rendered in the browser

---

If you add multiple figures of the object, the figures are displayed in a rotating carousel and are arranged in the order they are listed in the object information in `objects.yml`. If any of the object figures have a `caption` and/or `credit`, they will be included as a pop-up window. And if the

figure's `download` attribute is set to `true`, a download icon will be included as well.

> **TIP!**
>
> **In the table of object information, the items displayed and their titles are determined by the `object_display_order` attribute in the `objects.yml` file, as detailed in the section above. If the object information included a `link`, a "View in Collection" button is generated. The text of this button can be customized with the `objectLinkText` attribute in the project's `config.yml` file.**

Generate Object Lists/Grids

In a collection catalogue, there will typically be a visual table of contents for just the catalogue entries. To create a page with a list or visual grid of all the object entries, the entries themselves need to be grouped in their own section. In Quire, this means putting them in a subdirectory within the main `content` directory (Read more about it in the *Pages and Plain Text* page of this guide).

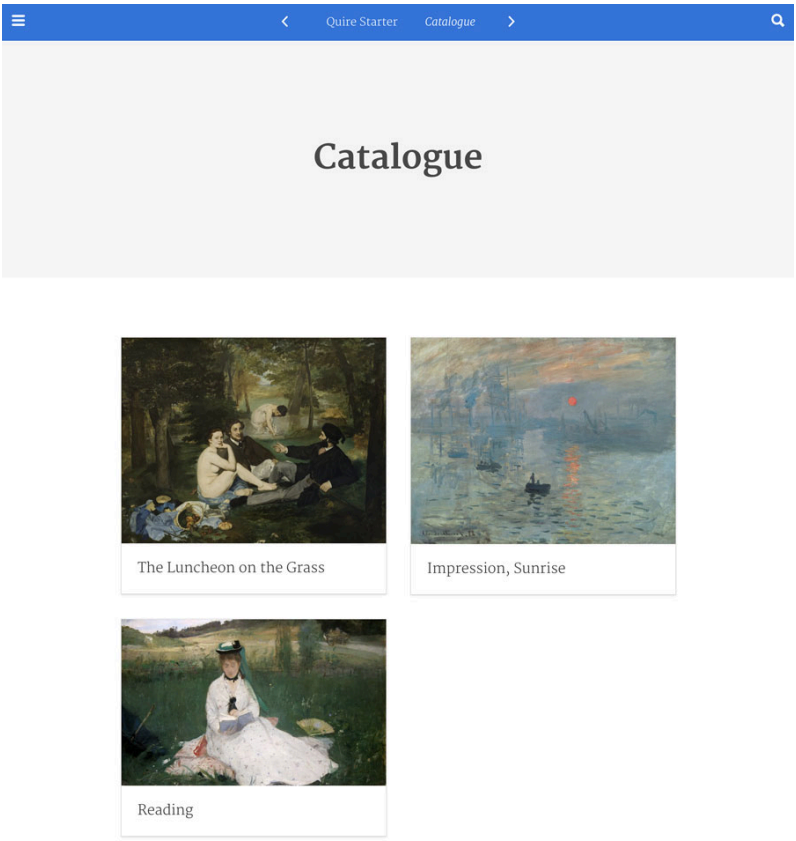
In this example, inside the `content` directory, we have a folder called `catalogue` and inside that, three numbered entries and an overview page:

```
catalogue
  overview.md
  1.md
  2.md
  3.md
```

The `overview.md` file is going to be our visual table of contents. To populate it, simply give it the attribute `type` a value of `"contents"` and

the attribute `class` a value of `"brief"`, `"list"`, `"abstract"`, or `"grid"` to determine the style. (The `"grid"` option will include an image from each entry page.) This `"contents"` page type will automatically generate from each of the Markdown files in the folder.

```
title: Catalogue
type: contents
class: grid
slug: .
```



screenshot of catalogue grid page as rendered in the browser

TIP!

The `slug` value in the sample above, will change the URL of the page. Instead of being `/catalogue/overview` it will be simply `/catalogue`. Read more about the function of `slug` in the *Pages and Plain Text* page of this guide.

Copyright & About Pages

Create Chicago and MLA formatted citations for your publication

Cite this Page

The Chicago and MLA formatted citations provided on every page of the online edition of a Quire publication are generated automatically based on data in your publication.yml file, and in the YAML of each Markdown page.

Key metadata about your publication that is included in publication.yml and will be used to generate your citations:

```
title:
subtitle:
reading_line:

series:
number_in_series:
```

```
pub_date: YYYY-MM-DD

identifier:
  url:

contributor:
- id:
  type:
  role:
  first_name:
  last_name:
  full_name:
```

Key metadata about individual pages/chapters included in the YAML of the specific Markdown page:

```
title:
subtitle:
contributor:
- id:
  first_name:
  last_name:
  full_name:
```

- For names, you need either first_name AND last_name, or just full_name
- Contributors identified as type: primary will be included in the citation as the publication's authors

- If your publication has one or more editors, rather than authors, they should be type: primary, and role: editor in the YAML.

For more information see the chapter on Contributors

Contributors

Credit and include multiple contributors

Quire is designed to credit and add contributors to publications in a flexible way. Contributors' data is stored in the `publication.yml` file of your project or in the YAML block of individual pages. The `q-contributor` shortcode offers multiple options to display contributors' data in your publication. As a reminder, a shortcode is a simple snippet of code inserted in a Markdown file that pulls in information from other files in your project.

Add Contributors to Your Project

Contributors can be listed under `contributor` in your publication.yml file, or, for contributors specific to a page in your project, in the YAML block at the top of that page.

At minimum, each contributor must have a `first_name` and `last_name`, or just `full_name`. In addition to these, wherever they are listed (publication.yml or pages YAML block), the following YAML attributes can be used for your contributors:

```
- id:
  type:
  role:
  first_name:
  last_name:
  full_name:
  file_as:
  title:
  affiliation:
  pic:
  url:
  bio:
```

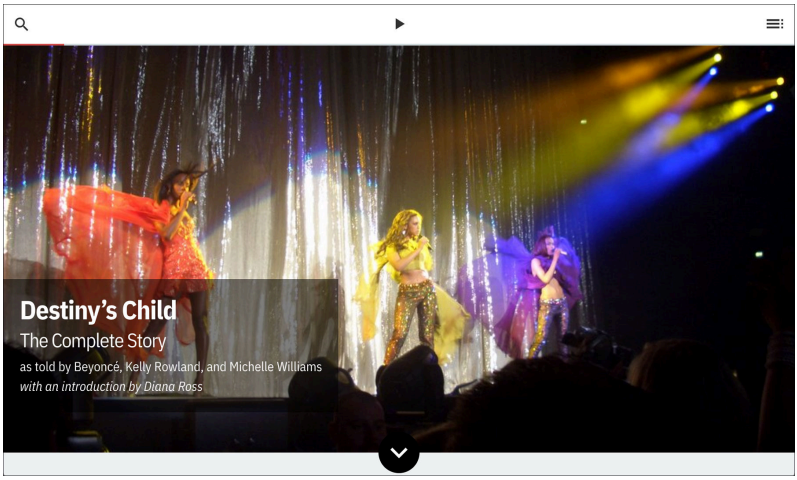
Display Contributors on Your Cover

Any contributor listed in your publication.yml file that has a `type: primary` will be considered a main author for your project and will be listed on the cover page, in the site menu, and in the metadata included in the project code. For publications with more than one author, names will be listed in the order they appear in the publication.yml file.

If the primary authors are editors of the book, you can specify that with `role: editor`.

Sometimes, rather than a plain list, you may want your contributors listed in a particular way. Such as, "Edited by Author Name and Author Name". In these cases, you can add your custom text in the publication.yml file under `contributor_as_it_appears` which can also take Markdown and HTML tags as needed.

```
contributor_as_it_appears: as told by Beyoncé, Kelly Rowland,
  and Michelle Williams <br /> *with an introduction by Diana
  Ross*
```



Sample publication cover with the contributors listed using the `contributor_as_it_appears` option in the publication.yml file, which allows for specific language and formatting to be applied.

While the `contributor_as_it_appears` value will override any contributor information otherwise listed, it is still recommended that you list the individual authors under the `contributor` area in your YAML, as this will be used as metadata for your book and will aid search engines and social media sites in discovering and listing your site.

Display Contributors on Individual Pages

Individual pages in your publication can have specific authors. Add them to the page YAML either with their names and other information, or by using an `id` that references a corresponding listing in your publication.yml file.

```
title: Introduction
type: page
contributor:
  - first_name: Kelly
    last_name: Roland
```

```
title: Introduction
type: page
contributor:
  - id: kroland
```

Contributors in the Page Heading

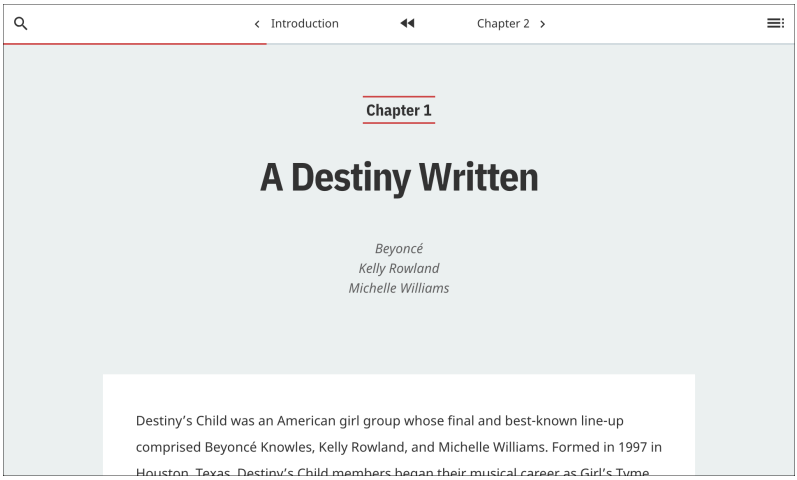
For most page types, you will see in previewing your site that contributors to a page will be automatically listed at the top of the page just under the title, in the order they appear in the YAML. By default, they will appear with their names and, if given, their titles and affiliations. You can override this by specifying a new value either on a page-by-page basis, or globally.

On an individual page, in the page YAML:

```
contributor_byline: name-title # name | name-title | false
```

For the entire publication, in the config.yml file:

```
contributorByline: name-title # name | name-title | false
```



By default, page contributors are listed in the page header, under the title. Contributor professional `title` and `affiliation` will be included unless `contributor_byline: false` is set in the page YAML, or `contributorByline: false` is set in your config.yml file.

While `name-title` is the default (and will be used if no value is specified), `name` will omit any title or affiliation information, and `false` will remove the contributor listing from the heading of the page altogether.

> **TIP!**
>
> **If you specify `contributorByline: false` you can still have names appear on individual pages by specifying either `contributor_byline: name-title` or `contributor_byline: name` on that page.**

Just as with the cover, if you want to display the contributors in a particular way, such as "Translated by Author Name", you can do so by specifying a `contributor_as_it_appears` value in the page YAML.

Contributors Elsewhere on the Page

You can also add lists of contributors to the main body of a page using the `q-contributor` shortcode. This allows you to create a page of contributor biographies, a section of bios for a single page, a list of contributors, a byline for a particular page, or other similar applications.

The shortcode requires both a `"range"` and a `"format"` value, and allows for an optional `"align"` value as well.

Sample: `{{< q-contributor range="page" format="bio" align="right" >}}`

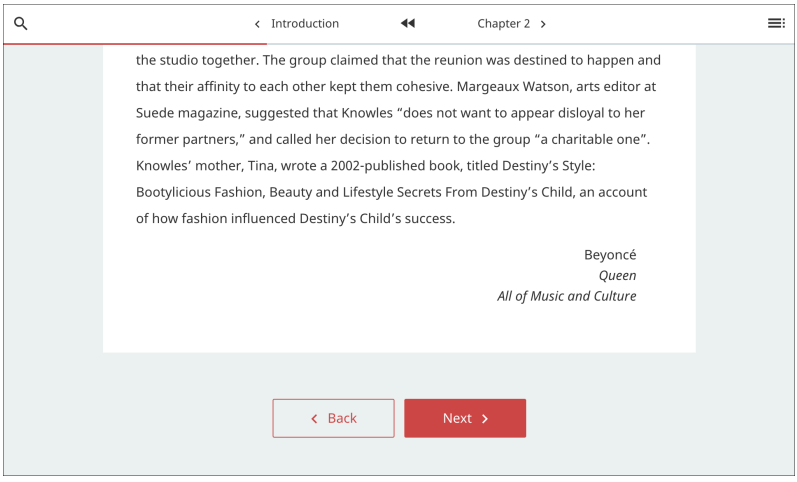
The `"range"` value determines which contributors will be included in the list. Predefined `"range"` values are:

Value	Description
`page`	Only the contributors listed for the page the shortcode appears on.
`all`	All contributors listed in the publication, whether listed on individual pages or in the publication.yml file.

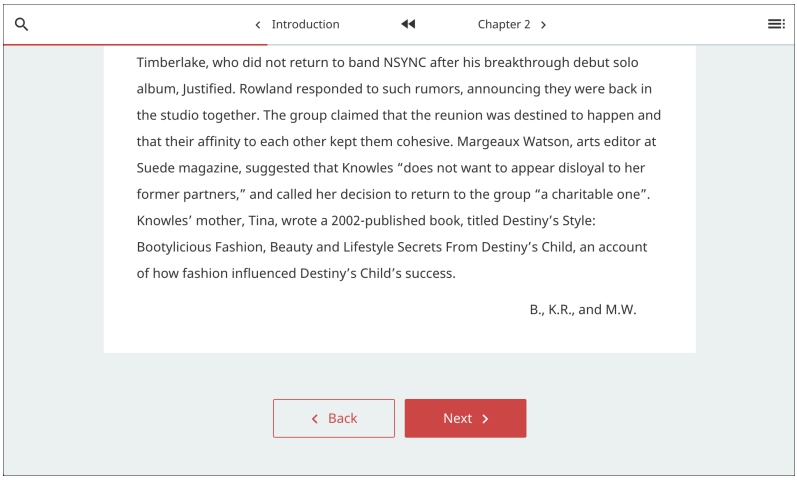
You can also use any contributor `type` you define. So if you give a contributor a `type: primary` (such as for your main publication authors, as discussed in the "Displaying Contributors on Your Cover"), then a shortcode using `range="primary"` will list any of your project's primary contributors.

The `"format"` value determines what information will be listed for each contributor in the `"range"`, and how it will be formatted. Possible `"format"` values are:

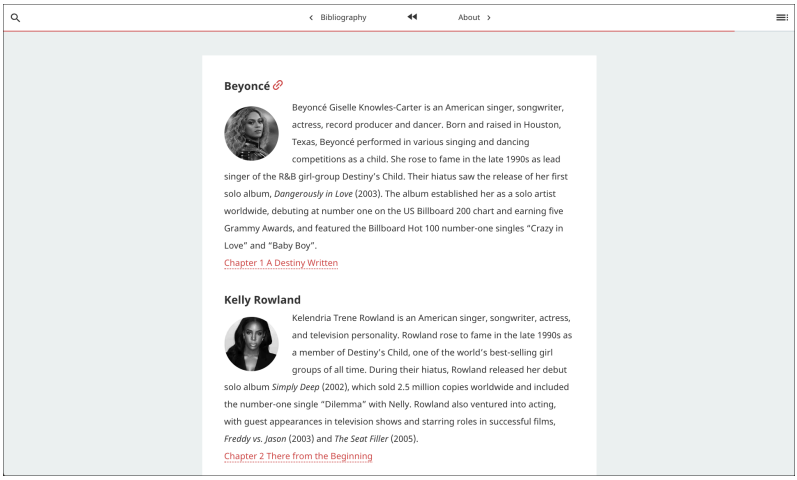
Value	Description
`initials`	Looks for the capital letters in a contributor first and last name and combines them together. Jane Pauley becomes J.P.; Ralph Waldo Emerson becomes R.W.E.
`name`	Just the name.
`name-title`	The name and, when available, the title and affiliation; on a single line
`name-title-block`	The name and, when available, the title and affiliation; broken onto separate lines.
`bio`	The name and, when available, a picture, offsite link to their personal site, and a bio. Plus links to any individual pages in the project for which they are listed as a contributor.



The `q-contributor` shortcode can be used to add a list of contributors anywhere on a page. In this case it is the page author formated with the `name-title-block` option.



Quire also includes an `initials` format which will list the author names onluy by initials.



The most full-featured way to list contirbutors is with the `bio` format. It includes the contributor's name, bio, picture, an offsite link to their personal site, and a link to any contributions they made to the publication.

The `"align"` value will align the text. If no value is given, text alignment will default to the left. The possible values are:

Value	Description
`left` (default)	Align the names and text to the left.
`center`	Align the names and text in the center.
`right`	Align the names and text to the right.

See the `q-contributor` shortcode reference for details on each of the standard contributor attributes.

Sort Contributor Lists

Using the shortcode, contributors will be listed alphabetically. Either by `last_name` `first_name` if given, or `full_name`. You can specify a `file_as` value for contributors to override the default sorting.

If you wanted, for example, a list of essay contributors ordered in the way they are ordered in the page YAML block, you could assign a numeric `file_as` value to each (1, 2, 3 etc.). Note, though, that this `file_as` override will carry over to other uses of the shortcode. For example, a complete list of contributors at the end of a volume of collected papers.

> **WARNING!**
>
> **Contributors with the same name will override each other and only one will appear, but using a `file_as` value would fix this. For example, if there are two Jane Smiths, assigning a `file_as` value of "Smith, Jane 1" to one and "Smith, Jane 2" will sort them in that order, but their names would still be listed as Jane Smith.**

Include Contributors for Search Engines

Contributor information is also embedded in Quire projects in a way that is optimized for search engine discovery. Here are a few tips to take advantage of this feature:

- List your project's main authors in the publication.yml file and give them a `type: primary`
- List other contributors (like authors of individual papers) in the publication.yml file and give them a `type: secondary`.
- Whenever using the `contributor_as_it_appears` value (which overrides how contributors are listed on the cover or on individual pages) still include a list of the individual contributors in your YAML. This is especially true for your overall publication, and any pages that have a `type: essay`, the metadata for which is structured to pay particular attention to the authors.

Style Customization

Alter look and feel with custom styles and themes

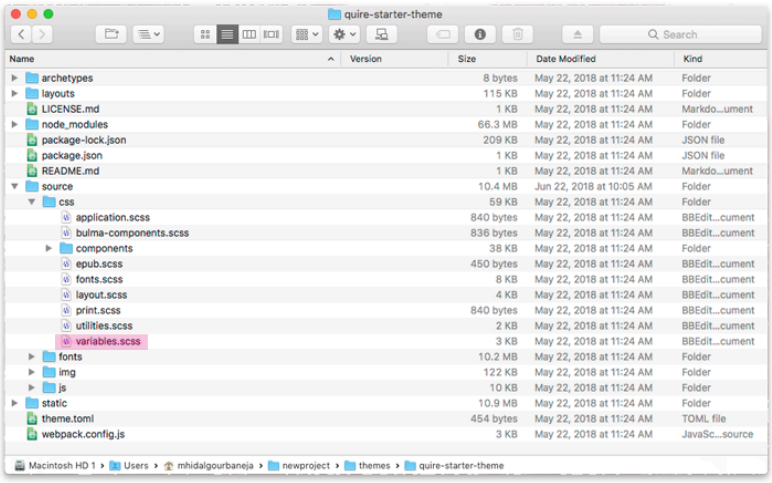
The look and feel of your Quire publication can be customized at four different levels of complexity:

1. Changing style variables in the theme
2. Adding new style rules to the `custom.css` file
3. Overriding specific theme templates with your own custom version
4. Creating an entirely new Quire theme.

Change the Theme Style Variables

When you first start a new Quire project by running the command `quire new`, a default theme is installed in the project's `themes` directory. This default theme includes a number of default style variables that will let you update text and background colors, some element sizes, fonts, paragraph indents, and more. This allows relatively easy customization without the need to dig into the stylesheets and CSS markup. A complete list of these variables includes information about the customizations available.

To find the variables, open the `themes/default` directory, navigate to the `source` sub-directory, and then `css`, and open the file called `variables.scss`.



screenshot of the `variables.scss` file in the default-theme directory

The variables are prefixed with a dollar sign and are descriptive of what they control. For instance `$quire-navbar-background-color` is the background color of the navigation bar at the top of every page. To make it red, you could enter:

```
$quire-navbar-background-color: red;
```

Colors are expressed a number of different ways, none of which are better or more supported than the others, so you can use your preference. Most common are:

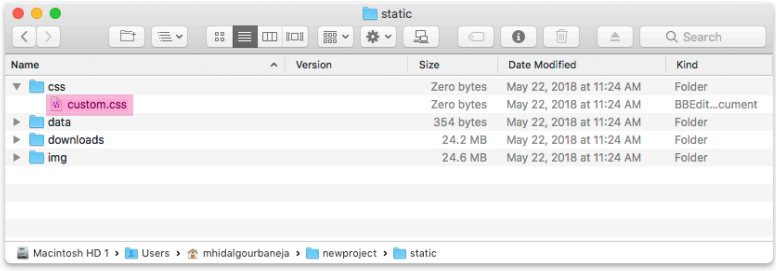
- The standard 140 Color Keywords such as `red`, `royalblue`, and `honeydew`

- The many possible HEX color values like `#FF0000`, `#4169E1`, and `#F0FFF0`
- RGB Color Values like `rgb(255, 0, 0)`, `rgb(65, 105, 225)`, and `rgb(240, 255, 240)`

> **WARNING!**
>
> **You must have the `quire preview` command running in your command-line interface to see changes you make to the `variables.scss` file. You may also need to refresh your browser page or clear the browser cache to get the style changes to fully load.**

Add Custom Styles

In your project's `static` directory, there is a `css` directory with a blank `custom.css` file.



screenshot of the custom.css file in the static directory

Any CSS you add here, will be added to your site's styles. For example, let's say you'd like a particular line of text in one of your Markdown files to be red. You could wrap that text in `<span>` HTML tags and give it a class.

```
<span class="red-text">This text should be red</span>
```

And then in your `custom.css` file, add a style rule for that class:

```
.red-text {
  color: red;
}
```

Custom CSS like this can also be used in conjunction with the `q-class` shortcode. While you can add `<span>` HTML tags within lines and paragraphs of text, in Markdown you can't do the same with `<div>` or `<section>` tags across multiple paragraphs. Instead, you can use the `q-class` shortcode to assign any class to all the Markdown within the opening and closing shortcode tags.

```
{{< q-class "red-text" >}}

This entire paragraph should be red.

As should the paragraph after it.

{{< /q-class >}}
```

Styles added to the `custom.css` file will also override any existing styles already in use in your theme. For example, the following option would apply the style to any element with a class of `"title"` anywhere in your publication.

```
.title {
  color: red;
}
```

To determine the selectors for any element in your browser of choice with your publication previewing, control-click (Mac) or right click (PC) on the element and select "Inspect element". This will show you the HTML markup for your site, along with all the class names and elements, and even the styles that are currently being applied to that element.

The more specific you can be with your CSS selectors, the more likely the style will only be applied to the specific element you want. For example, if you wanted the page title on a specific page to be a different color than the titles on the rest of the pages, you could determine the CSS selector for that element on that page and apply a style rule to it without changing the styles on any other element or page. This example limits the style to the title in the page header of that one page:

```
#chapter-one .quire-page__header .title {
  color: red;
}
```

In the above example, we are selecting the element with a class of `"title"` that is inside an element with the class of `"quire-page__header"` (both of which start with a period `.` to indicate class), that is inside an element (in this case an element representing the page itself) with an id of `"#chapter-one"` (which starts with a hashmark to indicate id).

TIP!

In Quire, page ids are unique, and can be found on the `<div>` element that has the class `"quire-primary"`. By using the id in your custom CSS, you are targeting only that page, not all `"quire-primary"` elements throughout your publication.

TIP!

Exceptionally, if somewhere there is a more specific CSS selector that's applying a style to an element, it will override the less specific one — even if it's in your `custom.css` file. If you are trying to apply a more global style change like this and you find it's not working, it may be because your CSS selector is too generic and there is a more specific rule elsewhere in your theme's styles that is overriding your more general one. The "Inspect element" tool will point to what combination of CSS selectors are actually applying the final style as it's seen in the browser window.

Override Theme Templates

CSS changes like those mentioned above are best for restyling existing elements. If you'd like to make a more structural change, say, to rearrange elements on the page, or add new elements altogether, you'll need to alter the template files that come in the theme. That said, other than changing the Variables in `variables.scss` file, as described above, it's usually best not to make other changes directly in the theme itself. By not doing so, it's easier to update your theme or switch out other themes later, not to mention easier to undo changes you've made.

For modest changes to the theme templates, we recommend creating new override files. Much like the `custom.css` file can be used to override styles in the project theme, you can also have files to override templates. There are none in the default starting project, so you'll need to start

creating a new `layouts` directory folder in the main directory of your project. Any files you put in this `layouts` directory will override the corresponding files in the `layouts` directory of your theme. This includes page templates, partial templates, and shortcodes.

For example, let's say you want to customize the layout of all the pages in your project with the `type` of `"essay"`. In the theme `layouts` directory you'll find that there's a sub-directory called `essay` with a file in it called `single.html`. This is the template that controls the `"essay"` pages. You can see there's a page header, an abstract, the main content `(.Content)` of the Markdown file, and a page bibliography.

```
{{ define "main" }}
<article class="quire-page" id="main" role="main">

  {{ partial "page-header.html" . }}

  {{ if .Params.abstract }}
  {{ partial "page-abstract.html" . }}
  {{ end }}

  <section id="content" class="section quire-page__content">
    <div class="container">
      <div class="content">
        {{ .Content }}
        {{ partial "page-bibliography.html" . }}
      </div>
    </div>
  </section>

</article>
{{ end }}
```

If you copy the `essay` subdirectory and its `single.html` file into the new `layouts` directory in your project's main directory, this copy will override anything in the theme. So, if you delete the bibliography and rearrange the header and abstract in the copied file, that's what Quire will use when building the site. It only changes the style of the header and abstract of your pages while the bibliography style remains intact.

```
{{ define "main" }}
<article class="quire-page" id="main" role="main">

  {{ if .Params.abstract }}
  {{ partial "page-abstract.html" . }}
  {{ end }}

  {{ partial "page-header.html" . }}

  <section id="content" class="section quire-page__content">
    <div class="container">
      <div class="content">
        {{ .Content }}
      </div>
    </div>
  </section>

</article>
{{ end }}
```

By default, Quire has a number of pre-defined page types like `"essay"`, `"entry"`, and `"cover"`. To create a new page type, you would follow the model of the `"essay"` page type above, and create a directory with the name of the type and in that, have a file called `single.html` with the template.

Whether in the theme or in your project directory, all shortcodes go in the `layouts` directory and `shortcodes` sub-directory. The name of the shortcode file corresponds to the way the shortcode is called in the Markdown files. So `q-figure.html` is the shortcode `{{< q-figure >}}`.

And if you make a mistake or change your mind later, you can simply delete the copy of the file and Quire will go back to using the original template as provided in the theme. This method can also be used to add completely new templates and even new shortcodes.

Default Theme Style Variables

Defined Variables

This default theme includes a number of style variables intended to allow for relatively easy customization without having to dig into the stylesheets and CSS markup. They are explained below and can be found and changed in `/source/css/variables.scss` file. Take special note of thee `$theme` variable which can be set to “modern” or “classic”, and the `$accent-color` variable which will give your publication a distinct feel in coloring all links, buttons and other navigation elements.

Variable Type	Variable	Expected Value	Description
Primary	$accent-color	color value	The color of buttons, links and navigation elements
	$content-background-color	color value	Main text area background color

Variable Type	Variable	Expected Value	Description
	$secondary-background-color	color value	Additional background color, only for modern version of the theme
	$theme	“modern”, “classic”	Shifts the overall feel and typography of the publication
	$navbar	“normal”, “accent”	Predefined options for your navbar, white or the accent color
Page typography	$quire-base-font-size	unit value	16px default
	$quire-page-paragraph-style	“line-space”, “indent”	Paragraph display styles, either a line space and no indent, or an indent and no line space
Cover typography	$quire-cover-text-color	color value	
	$quire-cover-text-scale	integer	Use decimal numbers such as .8 or 1.3 to shift the text larger or smaller
Menu colors	$quire-menu-color	color value	
	$quire-menu-text-color	color value	
Image viewer colors (pop-up figure	$quire-entry-image-color	color value	The background of the image viewer on entry pages

Variable Type	Variable	Expected Value	Description
viewer & entry page viewer)			
	$quire-entry-image-icons-color	color value	
Highlight color	$quire-hover-color	color value	Used on active links and when hovering over table rows
Print/PDF output	$print-width	unit value	8.5in default
	$print-height	unit value	11in default
	$print-bleed	unit value	.125in default
	$print-base-font-size	unit value	8.5pt default
	$print-text-color	color value	Specifying a plain black will avoid excessive color printing costs
	$print-splash-color	color value	Background color for full-bleed splash pages
	$print-entry-image-color	color value	Background color for full-bleed image pages
	$print-entry-image-display	“all”, “first”	Output “all” the images associated with an object on an entry page, or only the “first”, main image
Fonts	$quire-primary-font	font name	This theme includes three embedded, open license fonts, (Noto Sans, Noto Serif, and IBM Plex Sans Condensed) and also uses the widely available

Variable Type	Variable	Expected Value	Description
			Times. While other fonts can be specified for use, do so with caution and line spacing, element width and margin can all shift from font to font.
	$quire-headings-font	font name	See above
	$quire-footnotes-font	font name	See above
	$quire-navigation-font	font name	See above
Navbar	$quire-navbar-color	color value	
	$quire-navbar-hover-color	color value	
	$quire-navbar-text-color	color value	
Cover colors	$quire-cover-color-1	color value	The cover includes a gradient of two colors, specified here, and a white/transparent wave graphic specified in the cover.md of the content files
	$quire-cover-color-2	color value	

Variable Type	Variable	Expected Value	Description
Layout sizes	$quire-menu-width	unit value	352px default
	$navbar-height	unit value	3rem default
	$quire-entry-header-height	unit value	6rem default, determines the relative height of the entry page image viewer when in horizontal viewing mode
	$quire-map-height	unit value	500px default
	$quire-deepzoom-height	unit value	500px default

Special Classes

Class	Description
page-one	For PDF output, should be used on the page/chapter where you want page 1 to start, `class: page-one`, often an Introduction or first essay rather than the Contents pages or other frontmatter
backmatter	Can be applied to Markdown text with the `q-class` shortcode and text will be styled smaller, like the default footnotes style
is-pulled-left, is-pulled-right	Can be applied to figures or figure groups with their shortcodes, will make figures roughly half-column width and will float them to the left or right of the text

Class	Description
brief, list, abstract, grid	Can be applied to pages with a `type: contents`, will alter how the contents are displayed
side-by-side, landscape	Can be applied to pages with a `type: entry`, will alter how the image viewer is displayed

Configuration Parameters

Outside of the theme files themselves, a Quire project will also have a config.yml file which includes a number of parameters the theme relies on.

Parameter	Expected Value	Description
searchEnabled	boolean	Turn on or off the built-in text search capability for users
licenseIcons	boolean	Whether or not to display Creative Commons license icons
pageLabelDivider	string	". " default, etermines the text/ spacing to be inserted between page .label and page .title
citationPageLocationDivider	string	", " default, determines the text/ spacing to be inserted between the citation and the page number in the q-cite shortcode
displayBiblioShort	boolean	Whether a bibilography generated with the q-cite or q-bibliography shorcodes should display the short form of the reference, along with the long.

Parameter	Expected Value	Description
imageDir	string	"img" default, the directory in the `/static/` directory where you put your images
tocType	"full", "short"	"short" will hide all sub-section pages
menuType	"full", "short"	"short" will hide all sub-section pages
prevPageButtonText	string	"Back" default
nextPageButtonText	string	"Next" default
entryPageSideBySideLayout	boolean	Entry pages can have a side-by-side layout with image on the left and text on the right, this can be controlled by `class: side-by-side` in the page YAML, or globally with this parameter
entryPageObjectLinkText	string	"View in Collection" default
figureLabelLocation	"on-top", "below"	Whether the figure label is "on-top" of the image in the upper left corner, or "below" it with the caption
figureModal	boolean	If figures should be clickable to open into a full-screen modal window
figureModalIcons	boolean	Whether to display icons with the figure modal links
figureZoom	boolean	Whether figures should zoom or not inside the modal

Font Customization

Add external fonts to your publication

Typography is an important element of style in your Quire publication. Quire allows different levels of font customization, from using the already embedded open license fonts in the default Quire theme (`themes/default`), to adding new external fonts.

Customize Fonts

Quire's defualt theme includes three embedded, open license fonts: "Noto Sans", "Noto Serif", and "IBM Plex Sans Condensed". You can adjust which fonts are used where in the "variables" file of your theme, `themes/default/source/css/variables.scss` :

```
$ibm-sans: 'IBM Plex Sans Condensed', sans-serif;
$noto-sans: 'Noto Sans', sans-serif;
$noto-serif: 'Noto Serif', serif;

$quire-primary-font: null; // body and menu text
$quire-headings-font: null; // headings
```

```
$quire-footnotes-font: null; // footnotes and page backmatter
$quire-navigation-font: null; // navbar and next/prev buttons
```

The `$quire-primary-font`, `$quire-headings-font`, `$quire-footnotes-font`, and `$quire-navigation-font` variables are listed first with "null" values. Left as is, Quire will use the default fonts as specified elsewhere in the templates. You can choose your own combinations by using a defined font in place of "null":

```
$quire-headings-font: $noto-serif;
```

Add a New Font

WARNING!

Any font you add to your project should be under an open license, or you should have an explicit license to use it. While licensed fonts may offer variety, using them often means paying fees and tracking usage. Additionally, if you are using GitHub to publicly share your Quire project, licensed fonts should never be included in your repository without also being listed on your `.gitignore` file, as this will expose the files to other users.

For open license fonts, Google Fonts is a great source, but other more artisanal options abound, like the faces from the League of Moveable Type or even Cooper Hewitt's own open source font. For more free fonts and for thoughtful ideas about their use, Jeremiah Shoaf's *The Definitive Guide to Free Fonts* is worth the purchase price.

The steps to adding new fonts to your publication are:

1. Prepare Your Font Files and Add Them to Your Project

It's recommended to include your font files in multiple file formats in order to increase browser compatibility. Ideally, you will have each of your fonts in the following formats: `.eot`, `.woff2`, `.woff`, and `.ttf`. If this is not the case, you can use a free webfont generator, like the one from Font Squirrel, to produce these various formats from a single source.

All the fonts you'd like to add should go in a folder named after the font, and all should be named consistently. We recommend the following format with lowercase and no spaces:

```
cooper-hewitt
  cooper-hewitt-bold.eot
  cooper-hewitt-bold.ttf
  cooper-hewitt-bold.woff
  cooper-hewitt-bold.woff2
  cooper-hewitt-bolditalic.eot
  cooper-hewitt-bolditalic.ttf
  cooper-hewitt-bolditalic.woff
  cooper-hewitt-bolditalic.woff2
```

In your `themes/default` folder, all fonts are stored in `source/fonts`. Move your folder of fonts there.

```
cooper-hewitt
```

If you are using GitHub, and this is a licensed font, or a font you don't otherwise want available to anyone outside your project, add a line to your project's `.gitignore` file to make sure the fonts are not added to the git record.

You will continue to have the fonts available in your local copy of your project, but anyone working on a clone or fork of your repository will have to manually add your font files to their local copy for them to appear in the project properly when they preview or build the site.

When you ultimately host the final site on a web server, the fonts will be included in the built files and will need to be included in the package on the web server. Files hosted this way are not readily accessible to non-technical users, but are still public. For another layer of protection, font files could be assigned more generic names (ie., `f1-bld.ttf` instead of `cooper-hewitt-bold.ttf`). For complete protection of licensed/proprietary font files, other solutions should be sought.

2. Add Font Information to Your Stylesheets

Open the file `source/css/fonts.scss`. Each font in your font folder should have its own `@font-face` entry, as the examples in this file demonstrate.

```
@font-face {
  font-family: "Cooper Hewitt";
  src: url("../fonts/cooper-hewitt/cooper-hewitt-bold.eot");
  src: url("../fonts/cooper-hewitt/cooper-hewitt-bold.eot?#iefix
       url("../fonts/cooper-hewitt/cooper-hewitt-bold.woff2") fo
       url("../fonts/cooper-hewitt/cooper-hewitt-bold.woff") for
       url("../fonts/cooper-hewitt/cooper-hewitt-bold.ttf") form
  font-weight: 700;
  font-style: normal;
}
```

- The `font-family` name is what you will use to call the font in your stylesheets. It is typically in title case, and can include spaces. The `font-family` name should be the same for all weights and styles of

font you are adding. Meaning, all `""Cooper Hewitt""` not `""Cooper Hewitt" Bold"` or `""Cooper Hewitt" Light Italic"`.

The individual weights and styles are instead specified with the `font-weight` and `font-style` properties.

- The `font-weight` should be an integer set to match named weight of your font. Following the table below, a "Light" font would have a `font-weight` of 200. A "Bold" font would have a `font-weight` of 700.

`font-weight`	**Font name**
100	Extra Light or Ultra Light
200	Light or Thin
300	Book or Demi
400	Normal or Regular
500	Medium
600	Semibold, Demibold
700	Bold
800	Black, Extra Bold or Heavy
900	Extra Black, Fat, Poster or Ultra Black

- The `font-style` will be either `normal` or `italic`.

3. Use Your New Font

With the font files included in the `source/fonts` folder, and all the matching `@font-face` entries saved to the `source/css/fonts.scss` file, you can now use your font anywhere in your site CSS with `font-family`.

```
h1 {
  font-family: "Cooper Hewitt";
}
```

Typically, you'll want to change fonts across the project. For instance, making all the main body copy a new font, or all the headings. This can be done in the `source/css/variables.scss` file that we describe in the Customize fonts section:

To replace all `$sans-serif` uses with a new font:

```
$sans-serif: "Cooper Hewitt", Helvetica, sans-serif;
```

Or to leave the existing `$sans-serif` and just make all the primary font be our new one:

```
$family-primary: "Cooper Hewitt", Helvetica, sans-serif;
```

The rules about fallback fonts described in the Customize fonts section above also apply to the new fonts.

Additional Netlify Tips

Connecting Domains to Netlify

There 3 ways of connecting a domain to Netlify

- Purchase your domain through Netlify and run your DNS through their interface
- Add a proxy to your webserver
- Add an alias CNAME to your DNS to point to your Netlify domain

Any of these will work, it is more specific to what you want your domain to be.

Netlify Build Configuration

Instead of providing a `Production` directory or `Build` directory, you can create a `netlify.toml` file which will run commands from the root directory. These commands are set in the scripts block in the your `package.json` file. In Quire the path is `quire/themes/default/package.json` . Quire comes with a Netlify build command already, but as you will read below it is very easy to add or modify your own.

```
"scripts": {
 "build:netlify": "webpack --config webpack/webpack.config.prod.
}
```

This is the command we are running to build the Quire site in Netlify via our configuration below and comes installed with Quire. It first runs Webpack to build our assets, CSS, JS. Then it runs the Hugo command to build the static.

Now let's create the `netlify.toml` in the root directory. Copy and paste this text below into a new file called `netlfy.toml` and put it in the root directory of your project.

```
# netlify.toml
# The prefix is the path to your package.json file in your theme
# Change the path of your theme if it is not quire/themes/defaul

# This section is the production configuration and is all you ne

[build]
# Base is the path to your themes package.json file.
base = "themes/default"
command = "npm run build:netlify"

[context.production.environment]
HUGO_VERSION = "0.55.5"
HUGO_ENV = "production"
HUGO_ENABLEGITINFO = "true"
```

```
# These next configurations are optional

# This section is the pull request configuration
[context.deploy-preview]
# Base is the path to your themes package.json file.
base = "themes/default"
command = "npm run build:netlify"

[context.deploy-preview.environment]
HUGO_VERSION = "0.55.5"

# This section is the branch configuration
[context.branch-deploy]
# Base is the path to your themes package.json file.
base = "themes/default"
command = "npm run build:netlify"

[context.branch-deploy.environment]
HUGO_VERSION = "0.55.5"

# This section is the branch configuration but targets a specifi
[context.stage]
# Base is the path to your themes package.json file.
base = "themes/default"
command = "npm run build:stage"

[context.stage.environment]
HUGO_VERSION = "0.55.5"
```

Alter or add another command

When we run the build process on Netlify we may want to add flags to our Hugo command to make Hugo behave differently, either on a specific branch or in the preview deploy. Let's say, for example, we want to add the flag to build drafts for a branch and not for production. We have the command `npm run build:stage` above, let's use that.

Our scripts block will now be

```
"scripts": {
"build": "webpack --config webpack/webpack.config.prod.js && cd
"build:stage": "webpack --config webpack/webpack.config.prod.js
}
```

We are able to add the `-D` or `--buildDrafts` to the Hugo command to build drafts on our stage branch in our repository but not in master which can consider live or production.

Adding extra commands is a great way to preview code!

Private Submodules

Once you have a proper build and deploy on Netlify, you may need to have a private submodule to save content or images that you may not want to expose.

If you have a private submodule and need to tie it together with your project, there's a few steps you'll need to follow to get it linked and working.

If you're new to submodules, these are some useful guides:

- https://github.blog/2016-02-01-working-with-submodules/

- https://git-scm.com/book/en/v2/Git-Tools-Submodules

Once you have it working locally, you'll need to push it to Netlify.

- Go to https://www.netlify.com/support/ and fill out a ticket
- Request an SSH key for your private submodule
- Wait until they send you an SSH key
- Go to your deploy settings in your repo, on GitHub it's https://github.com/user/repo/settings/keys (swap user and repo with your account or org and the respective private submodule repo)
- Click 'Add deploy key', add name to save it as and paste your SSH key from Netlify

Once that's all in place redeploy and you should have your private submodule linking correctly. Go back to your Netlify deploy and trigger a redeploy.

Netlify and Git LFS

If you are using Git Large File Storage (LFS) to manage large files in your GitHub or GitLab repository, you will need to set two environment variables in the Netlify deploy settings for your site.

In Netlify, go to Site Settings > Build & Deploy > Environment. Click "Edit variables", add these two variables, and click save:

Key	Value
GIT_LFS_ENABLED	true
GIT_LFS_FETCH_INCLUDE	*.jpeg,*.jpg,*.png,*.epub,*.mobi,*.pdf

Modify the list of file types for GIT_LFS_FETCH_INCLUDE depending on what files you've previously chosen to have managed by Git LFS when you

originally set it up in your repository. If you're not sure, check your repository's `.gitattributes` file.

The settings will take effect on your next site deploy.

If they are not set, or set improperly, you will find that certain images may not show up on your site, and download files will not be available, or will download incompletely.

As an alternative, Netlify also offers it's own Large Media storage option, which you can read about at https://docs.netlify.com/large-media/overview/.

Output Your Project

Produce online, PDF, and E-Book versions of your publication

Quire is designed to create a website version, a PDF version and two e-book versions of your project from the same source files. Each can be customized in various ways as described below. Once your outputs are ready, visit the *Deploy Your Project* section of our documentation to learn how to deploy your project to the web.

Site Output

Create the HTML files for your project by running `quire site` in your command-line shell. The files will be built into your project's `site` folder along with all the necessary static assets like image files, stylesheets and script files. The `site` file will be updated and overwritten each time you run `quire site`.

TIP!

If you are including PDF and e-book downloads as part of your online site (this is Quire's default) you'll need to update those files by running `quire epub`, `quire mobi`, and `quire pdf` as described below before running `quire site`.

You can hide specific pages of your project from the site output by adding `online: false` to the page YAML. Or conversely, you can add pages exclusively to the site output, by adding `epub: false` and `pdf: false` to the page YAML and leaving the `online` attribute unset, or set to `true`.

Unlike `epub: false` and `pdf: false`, adding `online: false` does not stop a page from being built and included in the site output. Rather, it just unlinks those pages and adds a metadata tag to them to prevent them from being indexed by search engines. You may want to manually delete the pages from the `site` folder after outputting and before uploading to a web server if you want to be fully certain they can't be accessed online under any circumstances.

TIP!

Links to download the EPUB, MOBI, and PDF versions are automatically included in the sidebar menu of the online version of your project. These links can be removed or modified by making changes to the `resource_link` information in your `data/publication.yml` file.

You can add links to these files from anywhere in your markdown files by linking to `/downloads/output.epub`, `/downloads/output.mobi`, and `/downloads/output.pdf` respectively.

E-Book Output

Quire can output two different reflowable e-book formats: EPUB and MOBI. EPUB is the most widely used format and will work on most devices and for most e-book vendors. EPUB is an official specification of the World Wide Web Consortium (W3C), and Quire outputs the latest version: EPUB 3.2.

Very closely related to EPUB, the MOBI format is used almost exclusively by Amazon's Kindle. Making it available in your project will allow Kindle readers to load the e-book onto their devices directly. (Amazon itself asks

for EPUBs when selling/distributing through them, which it then converts to MOBI and other proprietary formats as needed.)

The MOBI file is derived directly from the EPUB version of your project, and does not have styling or customization options separate from the EPUB.

You can remove specific pages of your project from the e-book outputs by adding `epub: false` to the page YAML. Or conversely, you can add pages exclusively to the e-book outputs, by adding `online: false` and `pdf: false` to the page YAML and leaving the `epub` attribute unset, or set to `true`.

For developers creating custom templates, you can use templating logic to create specific outputs for the e-books by querying `if eq .Site.Data.params.epub true`. This parameter is set in the `config/epub.yml` file.

Create and View the E-Book Files

Create an EPUB of your project by running `quire epub` in your command-line shell. An `output.epub` file will be created and saved to your project's `static/downloads/` folder. This file will be updated and overwritten each time you run `quire epub`.

Create a MOBI of your project by running `quire mobi` in your command-line shell. An `output.mobi` file will be created and saved to your project's `static/downloads/` folder. This file will be updated and overwritten each time you run `quire mobi`.

EPUB files can be viewed on the default Books app on macOS, or on a number of free EPUB readers available for both Windows and Mac.

MOBI files can be viewed with Kindle Previewer on both Windows and Mac.

EPUBCheck Validation

If you will be distributing your EPUB file via e-book vendors/distributors, it will have to pass validation with EPUBCheck. EPUBCheck verifies that the file conforms to EPUB standards which ensures that it will work properly across devices. A valid EPUB will also ensure a valid MOBI file.

Quire's default output will pass EPUBCheck, but the EPUB standard is very strict and a number of things can lead to an invalid file. By far the most common errors are broken internal links in markdown files to other files or to heading or image anchors within the file.

While there is an online validator for smaller files (10MB or less) we recommend downloading EPUBCheck and using it directly.

1. Download and install Java from https://www.java.com/.
2. Download the ZIP file of the latest EPUBCheck release from https://github.com/w3c/epubcheck/releases.
3. Unzip the downloaded folder. Inside it is a epubcheck.jar file that you'll reference in the next step.
4. In Terminal or PowerShell Admin type: `java -jar path-to-epubcheck.jar path-to-output.epub`

EPUBCheck will output a list of any errors or warnings that exist in your file. Only the errors need to be addressed for the file to be considered valid by most e-book vendors. Warnings are optional. Errors will be referenced by filename and line number. The filenames will be internal EPUB naming and not correspond to anything in your markdown project files. See the tip below for looking inside the EPUB file to track down the source of these listed errors.

TIP!

Look inside an EPUB file by opening it in a text editor like Atom, or by manually changing the file suffix to ZIP and uncompressing the file. Just note that you can't/shouldn't make change to an EPUB file this way. Rather, make changes in the source markdown and YAML files of your project and re-output the EPUB file.

EPUB Styles

EPUBs in Quire have their own style sheet separate from any styles applied to the online version of your project. EPUB styles can be modified and added to in the `themes/default/source/css/epub.scss` file.

Quire uses Pandoc to create the EPUB files, and while consistent in outputting valid EPUB files, Pandoc also strips out a lot of the class names and other structures you may normally use to add custom styling. See the tip above for looking into the EPUB file, and use that to find class names and HTML structures as Pandoc outputs them for use as selectors in your CSS.

PDF/Print Output

Create a PDF of your project by running `quire pdf` in your command-line shell. An `output.pdf` file will be created and saved to your project's `static/downloads/` folder. This file will be updated and overwritten each time you run `quire pdf`.

By default, the PDF is output at full-resolution and with crop marks for professional printing. You may want to manually crop the pages and downsample the file to a smaller file size using a program like Adobe Acrobat if you are only making it available as a digital download.

Quire's PDF output is generated by PrinceXML which you should have installed when first installing Quire. PrinceXML is free to download for non-

commercial use, though it does add a logo watermark to the first page of the PDF ouput. A desktop license can be purchased that will remove the watermark and also allow for commercial use.

You can remove specific pages of your project from the PDF output by adding `pdf: false` to the page YAML. Or conversely, you can add pages exclusively to the PDF output, by adding `online: false` and `epub: false` to the page YAML and leaving the `pdf` attribute unset, or set to `true`.

For developers creating custom templates, you can use templating logic to create specific outputs for the PDF by querying `if eq .Site.Data.params.pdf true`. This parameter is set in the `config/pdf.yml` file.

Modifying and Styling the PDF

PrinceXML creates the PDF from the website version of your Quire site using CSS rules. You can modify Quire's PDF styles using CSS just like you would modify Quire's online styles. You can read more about styles in general in the *Style Customization* section of this guide.

There are a number of CSS variables defined in Quire that allow you to adjust various parts of the PDF output, including the page size. The default page size is 8.5 × 11 inches.

In the `themes/default/source/css/variable.scss` file, are a number of key print/PDF-related variables:

```
// Print/PDF stylesheet
// --------------------------------------------------------------
$print-width: 8.5in;
$print-height: 11in;
$print-bleed: .125in;
```

```
$print-base-font-size: 8.5pt;
$print-text-color: $black;

$print-splash-color: $off-white;

$print-entry-image-color: $rich-black;
$print-entry-caption-color: $white;
$print-entry-image-display: all; // first | all
```

There are even more PDF-related variables that can be modified in the `source/css/print.scss` file. Including page margins, and rules regarding the running feet and page numbering that are included at the bottom of each PDF page.

WARNING!

Beware of a couple known issues with variables:

In `source/css/print.scss` Quire defines both a `$print-base-text-column-width` as well as inner and outer margins. The problem with this is that if you add up the values of the two margins and the text column and they don't equal the value you have set as a print-width, you can get some unexpected results.

There is also some well-intentioned but ultimately flawed logic built in that can lead to unexpected results as some margins are automatically adjusted if the print width is less than or equal to 7" and again if it's less than or equal to 10".

Where a variable is not available, you can also add custom CSS to your `static/css/custom.css` file to achieve the desired result. You can target

changes to *only* the print output by wrapping your CSS rules in a media query.

For example, this would hide all `video` elements in the print output:

```
@media print {
  video {
    display: none;
  }
}
```

Some of the CSS used in styling the PDF is from the CSS Paged Media Specification. This is a set of CSS rules designed specifically to style things in a page-like manner, including controlling left and right page rules, page numbering, and running feet and heads. There is good information about this in PrinceXML's documentation.

Tips for PDF Design and Development

For developers and designers interested in making more extensive changes to the PDF output, you can make the process easier by using a PDF reader that will autoreload, and displaying a version of the PDF output in your browser.

When working on improving or modifying the styling of your PDF output, the basic methodology is to output the PDF with `quire pdf`, open it, look at what you want to change, write/modify your CSS accordingly, run `quire pdf` again, open it, look to see if the CSS change had the desired effect, rinse and repeat. It is, in a word, cumbersome. But there are some things that will make this process a little easier.

Use an Auto-Reloading PDF Reader

Adobe Acrobat (a popular PDF reader) won't reload the PDF you're looking at if the file has been changed. We recommend instead using a PDF reader, like Skim for macOS, that will reload the PDF every time it's

changed. For Quire development, this means you can open the PDF to a page you want to make a style change to, make the change in your project, run `quire pdf` and see that change happen as soon as the PDF process is finished running. It takes away the wasted time of closing PDFs, opening new versions and finding your place in them time and time again.

Display the PDF Version in a Browser

You can use your browser to display a decent, though not exact, preview of what the print output will be. It won't have the correct page sizes or margins and page numbering, but you'll see the overall text sizes and styles, figures, spacing between these elements, and other parts generally as they'll look in the PDF. This means that you can make changes to your CSS and see a live preview in the browser without having to output the PDF every time.

1. Open the `config.yml` file and at the bottom of the list of `params` temporarily add `pdf: true` , which tells Quire to build the PDF version of your site. Be sure to delete this line when you're done working on the PDF.

2. Open your site preview in Firefox or Chrome, it will look a little different and some elements may be missing or altered.

3. Right or Control-Click anywhere on the preview:

   Firefox: Select Inspect Element and then click the small page icon in the upper right of the window that opens. (On hover, the icon will say "Toggle print media simulation for the page".)

   Chrome: Select Inspect, click the three-dots menu icon in the upper right of the window that opens, select More Tools, and then Rendering. In the area that opens, scroll down to "Emulate CSS media type" and select "print".

You can also use the web inspector to help track down different HTML elements and CSS selectors that are effecting the final PDF output. This can make it easier to make changes that will have the desired effect.

Deploy Your Project

Learn various methods to preview & deploy your project online

A Quire site is designed to be hosted on virtually any web server, either one your institution already runs, or a new server from the hosting service of your choice. You do not need any special back-end setup. If you plan to include PDF and e-book files as part of you published project, it's important that you start by following the directions in the *Output Your Project* section of the documentation before proceeding.

Basic Deploy

When hosting a Quire site, you will typically follow these steps:

1. Update the `baseURL` in `config/site.yml` to match the URL where the site will ultimately be hosted.
2. Create the HTML files for your project by running `quire site` in your command-line shell. The files will be built into your project's `site` folder along with all the necessary static assets like image files, stylesheets and script files. The `site` file will be updated and overwritten each time you run `quire site`.
3. Upload the contents of the `site` folder to your web host based on the directions they provide.

If you do not already have a web server or hosting plan, we recommend using either Netlify or GitHub Pages.

> **TIP!**
>
> **Netlify's "Continuous Deployment" option and Github Pages require a basic understanding of how to create a repository, and commit and merge changes changes. If you are new to GitHub, we recommend starting with GitHub Docs to learn by topic or by visiting the *Project Management with GitHub* section of our documentation. We also encourage you to check out Coding Train's video series Git and Github for Poets.**

Netlify

Netlify enables you to create a quick preview site by using your project's `site` files or by connecting it with your Github account to generate a shareable preview site that automatically updates every time you push changes to GitHub. (Please note, while we use GitHub and reference it throughout our documentation, you can also link Netlify to your GitLab or BitBucket account.) You can also use Netlify to host your final project when it's ready to publish.

To learn more about connecting domains, build configuration, private submodules, and using Git Large File Storage with Netlify, see *Additional Netlify Tips*.

To get started, sign up for a Netlify account and, optionally, connect it with your GitHub account.

Manual Deployment with Netlify

Manual deploy is ideal if you have a small site or want to run a quick preview. You can also use this option without needing a GitHub account. However, each time you make an update, you will need to go through the process of rebuilding the site, compressing files, and reuploading them to

Netlify, which may be burdensome if you have a lot of images or larger files. For continuous deployment please see *Continuous Deployment with Netlify*.

1. When you are ready to launch your project, run the `quire site` command in your command-line shell.
2. Navigate to your project in your home directory and compress the `site` folder.
3. Go to Netlify Drop: https://app.netlify.com/drop. Make sure you are logged in to your account and then drag-and-drop your compressed `site` folder into the indicated area.
4. You will be given a default URL to preview your project. Rename this URL by navigating to "Site settings" and changing the site name. (You also have the option to buy a domain or set-up a domain you already own.)

If you make further edits to your project and would like to preview them you will need to repeat this process.

1. Delete the old compressed `site` folder.
2. Run `quire site` again (your files will be automatically overwritten.)
3. Compress the newly updated `site` folder.
4. In Netlify, navigate to "Deploys" at the top of the page. You will see a blank space that reads, "Need to update your site." Simply drag-and-drop your new compressed `site` folder here and your link will be automatically updated.

WARNING!

One downside to manual deploys is that they can sometimes get stuck while uploading. To ensure a reliable deploy, use this option for Quire sites under 50MB and avoid individual files over 10MB. It's also recommended that you use the latest

version of the Chrome browser. For more tips visit the **Netlify Support Forum**.

Continuous Deployment with Netlify

By keeping your project files on GitHub and linking them directly to your Netlify account, any time you merge changes in Github, your preview link will be automatically updated. This process requires a few extra steps to get set-up, but will save time in the long run.

1. If you haven't already, create a repository for your project on GitHub.
2. To proceed with deployment, you will need to add two files to your repository: `netlify.toml` and `package.json`. Download them, add them to your repo, and merge changes before proceeding.
3. Log in to Netlify. On the "Team Overview" page, click the button that says "New Site from Git."
4. Next you'll want to "Connect to Git Provider."
5. If you see "no repositories found" you will be prompted to configure Netlify on GitHub.
6. Once this configuration is complete, choose the repository you would like to preview.
7. Set the "build command" as `npm run build` and the "publish directory" as `site`.
8. Hit "Deploy Site." Depending on the size of your project, this may take a few moments. Follow along with the build process (and check for errors) by navigating to "Publication Deploys" section and scrolling down to the "Deploy Log".

9. You will be given a default URL to preview your project. You can rename this URL by navigating to "Site settings" and changing the site name. (You also have the option to buy a domain or set-up a domain you already own.)

10. Should you need to make any updates to your site, just merge the changes and Netlify will automatically update your preview link. You can check "Production Deploys" in the site overview section to these track changes.

TIP!

Now that you have linked Netlify to your Github account, you'll see notifications about Netlify testing the site each time you submit a new pull request. If the checks pass, you can click the bottom-most link to launch a preview of your site. If the checks fail, there may be broken links, incorrect YAML, or other issues with your project files.

GitHub Pages

GitHub enables you to not only host your project code, but you can also use it to host a live version of your site. Learn more on the GitHub Pages website.

Preview Your Project with Github Pages

1. First, if you haven't already, follow the steps to host your project's code on GitHub in the *GitHub* section of our documentation to create a repository for your project.

2. In your text editor, open the `config/environments/github.yml` file.

3. Update the `baseURL` to correspond to your own GitHub site. It will follow the pattern: https://YOUR-USERNAME.github.io/YOUR-

PROJECT-DIRECTORY-NAME. So, if your GitHub username is bonjovi and your project file is blaze-of-glory, your `baseURL` would be https://bonjovi.github.io/blaze-of-glory.

4. Set the `canonifyURLs` to `true`.
5. Next, navigate to `themes/default/webpack/webpack.config.prod.js`.
6. So that the site fonts display properly, change line 80 to `outputPath: "[YOUR-PROJECT-DIRECTORY-NAME]/img/"` And change line 92 to `outputPath: "[YOUR-PROJECT-DIRECTORY-NAME]/fonts/"`.
7. Commit and merge these changes to the repo.
8. Open Terminal, navigate to your project and enter:

```
bin/deploy.sh
```

   This runs a script to deploy your site to GitHub pages. The script may ask for your GitHub username and password. (Remember that the password cursor won't move to show that you're typing. Just type the password and hit enter.)

9. If your repository is private, you will need to navigate to your repository on GitHub and update your settings.
10. In "Settings" scroll down to the "Danger Zone" and click on "Change project visibility."
11. For "Source" switch the branch to `gh-pages` and save.
12. Your site should now be published at https://YOUR-USERNAME.github.io/YOUR-PROJECT-DIRECTORY-NAME.

Accessibility Principles

Learn about WCAG 2.0 principles guiding Quire's development

As a publishing tool, Quire's goal is to maintain accessibility for keyboard and screen reader navigation, as well as for devices and browsers of varying sizes and capabilities and with limited functionality, such as those operating without JavaScript and/or CSS.

The principles outlined below have been informed in particular by:

- The 18F Accessibility Guide
- Carnegie Museums of Pittsburgh Web Accessibility Guidelines
- *Adaptive Web Design*, by @AaronGustafson
  https://adaptivewebdesign.info
- *Inclusive Design Patterns*, by @heydonworks

While not exhaustive, the list below is meant to highlight the *key* principles by which Quire was originally developed and that we recommend others follow when developing their own Quire projects. It has been ordered roughly, starting with those items most owned or effected by editors working on publication content and progressing into those owned or effected by developers working on publication styles, template markup, and interaction.

Key Principles

- Heading levels (`H1` through `H6`) should indicate a content outline, not visual styles, as they are frequently used by screen readers for page navigation. Quire pages will have their titles placed in an `H1` tag at the template level, and there should only ever be one `H1` tag on a page. Headings in the Markdown content documents should start with `H2` . All headings should have content following them.
- All non-decorative images should have `alt` descriptive text. (Tips on crafting good alt text.)
- All formats (PDF, EPUB, MOBI, and print) must offer at least basic access to *all* the content of the publication. For example: videos, deep zoom images and maps should appear with fallback images; URLs to online content should be provided in text, and hover-over citations or glossary terms should be printed in full at the bottom of the page or in a separate section.
- A proper background/foreground color contrast ratio must be maintained for all elements. (Color contrast checker.)
- Links must have a visual indicator besides only color.
- All page content should be in an ARIA Landmark element/role.
- The first element on every page should be a "Skip to Main Content" skip link.
- Any element or piece of information that inherits meaning or context from its *visual* appearance, should be augmented with spoken, descriptive labels for screen readers. Quire templates make use of a `visually-hidden` CSS class. When applied to an element, the text within is hidden from view, but will be read aloud by screen-readers.
- If clicking on an interface element sends the user to a new page or a new location on the existing page, it should be an `<a>` link. If clicking changes the state of the current page, such as in opening a modal, it should be a `<button>` .

- Buttons that hide/reveal content and rely on JavaScript to do so, should be progressively created with JavaScript on the client side. In this way, if JavaScript is disabled or not functioning, the user will have access to *all* the content of the page.
- When viewed without JS, the page should be beautifully designed and fully navigable with no missing/hidden elements.
- When viewed without CSS, the complete contents of the page must be logically organized and readable.

Glossary

Key terms and definitions related to Quire

A B C D E F G H I J K L M N O

P Q R S T U V W X Y Z

ATX-style Markdown headers: It is the type of Markwdown headers that are used in Quire. It consists of one to six # signs and a line of text. For more details visit the Pandoc manual: https://pandoc.org/MANUAL.html#headers

BISAC Subject Codes: Developed by BISG, the BISAC Subject Codes List, or BISAC Subject Headings List, is a standard used to categorize books based on topical content. For more information visit: https://bisg.org/page/BISACSubjectCodes

Blackfriday: Blackfriday is the Markdown processor used by Hugo to render the formatted text of the publication. You can find technical details at: https://github.com/russross/blackfriday

Bulma variables: Sass variables used by Bulma, an open source CSS framework. You can check a list of Bulma variables at: https://bulma.io/documentation/overview/variables/

Color Keywords: Color keywords are case-insensitive identifiers that represent specific colors. A comprehensive list of color keywords is at: https://www.w3schools.com/colors/colors_names.asp

command-line interface: A command-line interface (CLI) is a text-based user interface for running program tasks. In it, the user issues commands in the form of typed strings of text. This is opposed to a graphical user interface (GUI) for a software program, in which the user controls the program through visualized buttons, toggles and menus. https://www.codecademy.com/articles/command-line-interface

command-line shell: The command-line shell is a text-based window into the contents of your computer, and a space where you can run program commands. On Mac computers you'll most often use a shell called Terminal, on PCs it's PowerShell or Git BASH.

Copyright: Copyright is a form of protection granted by law for original works of authorship fixed in a tangible medium of expression. For more information about U.S. copyright visit: https://www.copyright.gov/help/faq/faq-general.html#what

Creative Commons Licenses: Creative Commons licenses are public copyright licenses that enable the free distribution of a work. Every license helps creators retain copyright while allowing others to copy, distribute, and make some non-commercial uses of their work. You can check the different types of Creative Commons licenses at: https://creativecommons.org/about/cclicenses/

class: The CSS class selector selects elements with a specific class attribute. The selector starts with a period `.`

CSS: Cascading Style Sheets or CSS is a stylesheet language used to define sytles and layouts for webpages written in HTML. For a deeper dive, Mozilla Developers provides a good guide to CSS: https://developer.mozilla.org/en-US/docs/Web/CSS

CSS selector: CSS selectors are used to target the HTML elements on web pages. There are several CSS selectors available, being one of these the CSS Class Selector.

DOI: The digital object identifier (DOI) is a unique alphanumeric string that identifies content and provides a persistent link to its location on the Web. DOIs are assigned the International DOI Foundation: https://www.doi.org/

element: An HTML element is an individual component of an HTML document or web page. Elements are defined by tags.

Free webfont generator: A tool that generates copies of your font in multiple formats for display on the web. A good webfont generator can be found at: https://www.fontsquirrel.com/tools/webfont-generator.

Getty Vocabularies: Controlled vocabularies developed to ensure consistency in cataloging and more efficient retrieval of information. For more information visit: https://www.getty.edu/research/tools/vocabularies/

GitHub: GitHub is a code hosting platform for version control and collaboration. For an introductory guide to it, visit: https://guides.github.com/activities/hello-world/#what

Git project management tool: Project management tool for software development that uses Git as their version control language. There are multiple tools available in the web.

HEX color value: Hexadecimal color values that represent specific colors. Colors are specified by hexadecimal integers between 00 and FF. For more information visit: https://www.color-hex.com/

home directory: A home directory is a file system directory on a multi-user operating system that serves as the repository for a user's personal files, applications, music, movies, downloads, etc. The directory name usually defaults to the system user's name. You can find the directory by

opening your Finder or File Explorer window and typing Command-Shift-H.

Hugo: The static site generator that powers Quire: https://gohugo.io/

JSON: JSON (JavaScript Object Notation) is a human-readable, open-standard file format written with JavaScript. It is used to store and exchange data through value types like strings, numbers, booleans, lists, objects, and null.

IANA: Internet Assigned Numbers Authority (IANA) is responsible for coordinating Internet's globally unique identifiers. DNS Root, IP addressing, and other Internet protocol resources are performed as the Internet Assigned Numbers Authority (IANA) functions. For more information visit:https://www.iana.org/

id: In an HTML document, the CSS ID selector matches an element based on the value of its id attribute.

Image Optimization: Set of techniques that compress image for the web. For more information about web image sizing and optimization, visit Google's Web Fundamentals guide on Image Optimization https://web.dev/fast/#optimize-your-images.

ISBN: The International Standard Book Number, or ISBN, identifies books or book-like products, as well as their publishers. There is one ISBN agency per country, the U.S. ISBN Agency can be found at: https://www.isbn.org/

ISO 8601 format: ISO 8601 describes an internationally accepted way to represent dates and times using numbers. More information about the format can be found at the ISO website: https://www.iso.org/iso-8601-date-and-time-format.html

ISO 639-1 language code: ISO 639 is the International Standard for language codes. More information about the standard can be found at the ISO website: https://www.iso.org/iso-639-language-codes.html

ISSN: The International Standard Serial Number, or ISSN, identifies newspapers, journals, magazines and periodicals of all kinds and on all media–print and electronic. For more information visit: https://www.issn.org/

Markdown: Markdown is a text formatting standard that defines the use of very simple text character combinations in order to indicate structure and formatting that can easily be transferred to more complicated HTML (web markup). For example, something surrounded in asterisks in Markdown turns into italics in the final publication: `*emphasis*` = *emphasis*.

Markdown file: A file formatted in Markdown that generates a Web page. Markdown files have a YAML block at the top containing the metadata of the page.

Markdown Processor: A Markdown processor parses Markdown code and generates the formatted text we get on the screen.

Media query: A media query is a method in CSS of applying styles to only specific media. For instance, only to print, or only to screens of a certain minimum or maxium width such as in responsive design. Read more in https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Open Access: Open access (OA) refers to free, unrestricted online access to content, frequently research outputs.

Pandoc: An open source, command-line text conversion tool that is included as part of the basic Quire installation. You can visit the Pandoc Demos page that lists the commands for the most frequent types of file conversion.

RGB Color Values: The RGB model define colors according to their red, green, and blue components by using hexadecimal and functional notations. An RGB color value calculator can be found at: https://www.w3schools.com/colors/colors_rgb.asp

Reflowable: A "reflowable" format is one where the text and images are not fixed in static layout like that of a printed book, but rather reflow or rearrange depending on the type and size device they are viewed on. Both web browsers and e-book readers are considered reflowable mediums.

Search Engine Optimization (SEO): Search engine optimization, SEO, is the process of increasing the online visibility of a website or a web page in a web search engine's unpaid results.

Static site Generator: A static site generator is a framework that generates a static website from source files.

Shortcode: A shortcode is a simple snippet of code inserted in a Markdown file that pulls in information from other files in your project. In Quire, shortcodes are used for styling, figure images, citations, bibliographies, and collaborators. Quire suports a range of shortcodes, and custom shortcodes can also be added.

Theme: Themes define the overall style of your website. It determines the use of colors, layout elements, and text positioning.

Template: Themes may include one or more templates. Templates are the variety of layouts a theme has.

Terminal: Terminal is the Bash Command-line Interface for macOS. An introductory lesson to the Basch Command-line can be found at the Programming Historian: https://programminghistorian.org/en/lessons/intro-to-bash.

Text editor: A text editor is an application used to edit text files containing either plain text or markup for rich text. All computers have basic text editors preinstalled, though we recommend using a free, more fully featured option like Atom or Visual Studio Code.

two-factor authentication: Two-factor authentication provides an extra layer of protection ensuring the security of your online accounts. A computer user is granted access only after providing two pieces of

information (or factors) -your password and another identifying factor, usually a 6-digit code generated through an authentication app.

Unicode: Character encoding standard that provides a unique number for every character allowing data to be transported through different platforms, devices and applications without corruption. For more information visit: https://unicode.org/.

UUID: A universally unique identifier or UUID is a 128-bit number used to identify information in computer systems

Web browser: A web browser is a program for browsing the web, but can also be used for viewing HTML and other web files directly from your computer, inlcuding previews of Quire projects.

Variable: CSS variables are entities defined by CSS authors that contain specific values to be reused throughout a document. For more information about CSS variables visit: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties.

Version control: Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Git is the version control system that Github host. You can learn more about version control at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

YAML: YAML is a plain-text, human-readable format for writing and storing data. YAML can be used in a standalone file with the file suffix `yml`, or inside a Markdown (`.md`) file. Read more in the *Fundamentals: YAML & Markdown* chapter of this guide.