

BETA

L

QUIRE

BETA

BETA

Quire

Documentation Site for the Getty's Multiformat Publishing Framework

[Getty Publications](#)

Quire is currently in closed development. To request access, please fill out our Quire Beta Request form.

Quire is a modern, multiformat publishing framework 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 ever signing-up for an account, paying a fee, or setting up and maintaining a complicated server.

Quire books are scholarly, visual, and built for discoverability and longevity. They feature:

- Page-level citation
- Footnotes
- Bibliographies

BETA

- Figure images and image groups
- Zooming images and maps
- Video and audio embeds
- Dynamic tables of contents and menus
- Full-text search
- Responsive web design
- Web accessibility and SEO optimized
- Outputs to web, PDF, and EPUB formats

View some of the books developed with early versions of Quire.

For more information, contact Greg Albers, Digital Publications Manager at the J. Paul Getty Trust, at galbers@getty.edu.

Contents

Quire: The Getty’s Multiformat Publishing Framework	<i>i</i>
Tutorial: Quire Basics: A Step-by-Step Walkthrough of the Fundamental Parts of Quire	<i>vii</i>
GUIDE	
Install or Update	1
Getting Started	7
Fundamentals: YAML & Markdown	12
Metadata & Configuration	18
Pages	22
Collection Catalogues	28
Figure Images	32
Zooming Images & Maps	37
Citations and Bibliographies	38
Copyright & About Pages	41
Contributors	42
Styles Customization	45
Fonts Customization	50

BETA

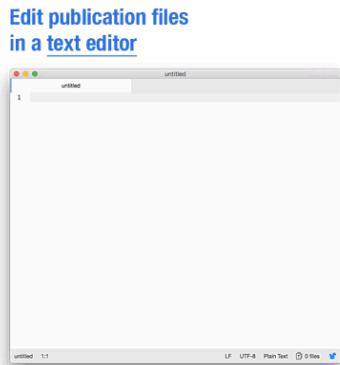
Multiformat Output	54
API-DOCS	
YAML	55
Shortcodes	63
Dependency Guide	68
RESOURCES	
Quire Cheatsheet	72
Default Styleguide	74
Quire CLI Instructions	77
GitHub for Quire	78
Sample <code>publication.yml</code> File	80
Accessibility Principals	82
About	84

Tutorial: Quire Basics

A Step-by-Step Walkthrough of the Fundamental Parts of Quire

- | | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">1. Working in a Command-Line Shell2. Installing Quire3. Creating a New Project4. Working in a Text Editor5. Entering Publication Metadata | <ul style="list-style-type: none">6. Editing Content7. Customizing Styles8. Outputting Your PublicationCongratulations! Now What? |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Quire is a publishing *framework*, meaning it's not one tool or process, but rather a network of many things hooked together. When you work in Quire, you'll be using a **text editor**, a **command-line shell**, and a **web browser**. You use the text editor to edit your publication files, the command-line shell to tell Quire what to do (like `quire new` to start a new project and `quire pdf` to build the PDF version), and the web browser to preview your work.



When you work in Quire, you'll be using a text editor (left), a command-line shell (center), and a web browser (right).

In the following sections, we'll get more familiar with these tools, and get you up and running in a demo Quire project.

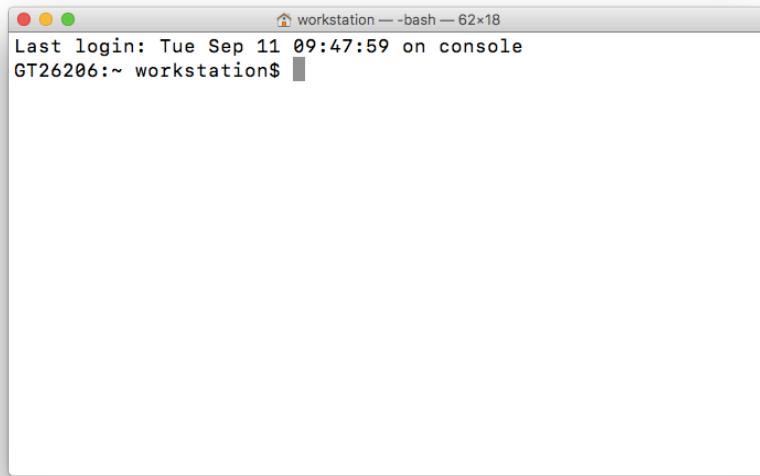
1. Working in a Command-Line Shell

The first thing you'll need is a command-line shell. Along with using it to run Quire, we'll also use it to install some of Quire's dependencies (the other programs Quire is dependent on in order to run such as Hugo to create the online site, and Pandoc to create the e-book files).

Macs already have a good shell installed. It's called Terminal and it can be found in the Applications/Utilities folder. (Or press Command-Space Bar and type "Terminal" to search for it.) For PCs, we recommend installing Git for Windows which comes with a shell called Git BASH.

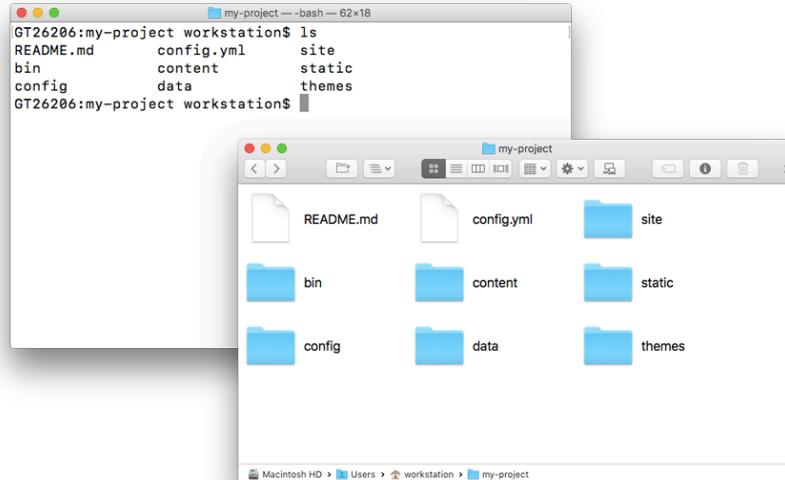
The shell is a text-based window into the contents of your computer, and a space where you can run program commands. Like opening a Finder or File Explorer window on your computer that shows the contents of a particular folder (directory), when you open your shell, you will also be in a particular directory. Your main user directory by default.

BETA



When you first open your command-line shell, you will be in your main user/home directory by default. In this case, “workstation”.

With the shell open, you can type `ls` (list) to list the folders and files in your current location. Or type `cd` (change directory) and the name of one of those other folders (like `cd Downloads`), and the shell will take you into it.



Both the command-line shell (left) and the Finder window (right) are views of the same directory on your computer. In this case, we are in a directory called “my-project” in a user account “workstation” and we can see a number of files and sub-directories which happen to make up a Quire project.

Command-Line Quick Reference



`ls` lists all the files in the directory you're in

`cd` followed by a space and a directory name, will move you into that directory: `cd my-project`

`cd` by itself will return you to your Home directory

`!!` will re-run the last command you entered

Pressing Control-C will stop any process running

For a deeper dive into the command line, check out a [Really Friendly Command Line Intro](#), or the Programming Historian's "Introduction to the Bash Command Line".

2. Installing Quire

Follow the links below to install Quire:

- MacOS
- Windows
- Linux

3. Creating a New Project

To start a new Quire project, open your command-line shell and type `quire new my-project`. Quire will download a new starter project named “my-project” into your current directory. If you are using the Beta, you may need to enter your GitHub username and password twice during the download process: once for the starter kit and again for the starter theme.



You can call your project anything you want, it doesn't have to be `my-project`, but don't use spaces, and we recommend lowercase.

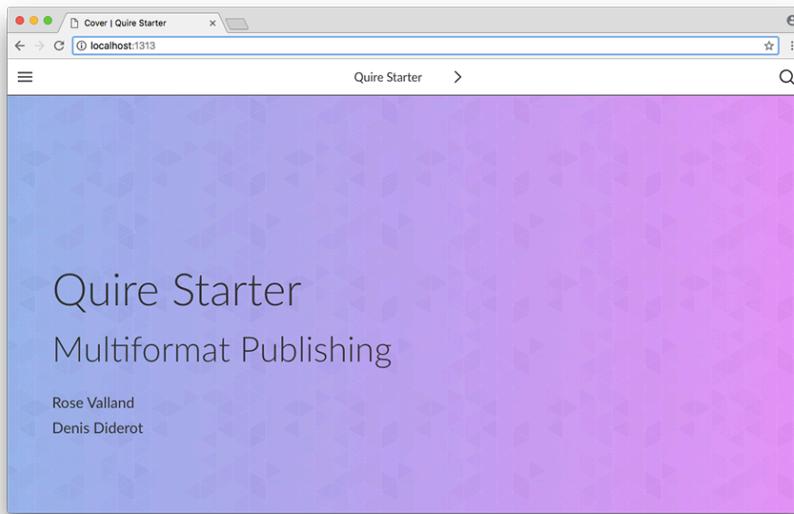
Once the process is complete, still in your shell, type `cd my-project` and press enter (which means change directory into the directory called “my-project”, which was just created). Next type `quire preview` and press enter again.

```
cd my-project
```

BETA

```
quire preview
```

To see the preview of your new starter, open your browser and go to <http://localhost:1313>.



When you run `quire new` Quire starts you with a demo project with some default content. You can edit, delete and add from there in building your own publication.

4. Working in a Text Editor

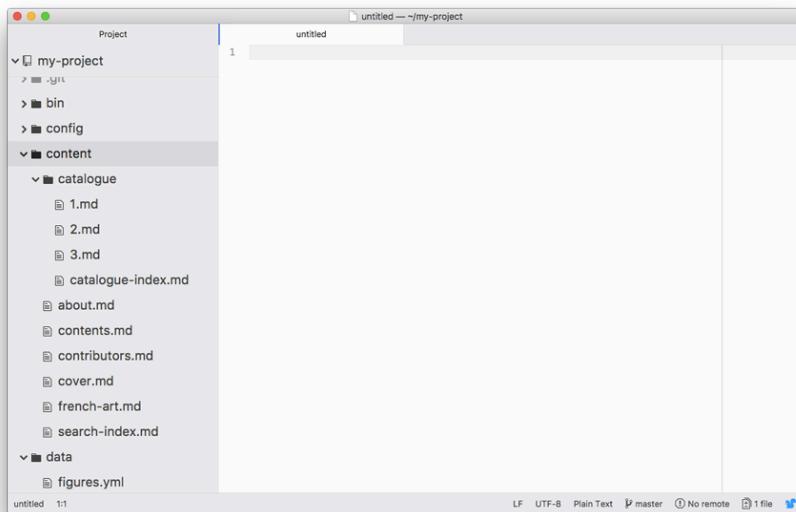
Some placeholder content comes with each new Quire project. To start customizing it, you'll need a text editor. This goes with the command-line shell you have for installing and entering Quire commands, and the browser you use for previewing.

Like its name implies, a text editor is simply a program to edit text. It's like Microsoft Word, but instead of dealing with text formatted for print, text editors specialize in text formatted for code and markup. You can use a text editor to edit all the different kinds of files in your Quire project.

Macs and PCs come with some simple text editors built in, but we recommend using one that offers more in terms of autoformatting as well as being able to see and work in multiple text files at a time. Download and install one of these good, free text editors:

- Atom
- Visual Studio Code

Once installed, open your text editor and locate and open the `my-project` directory you created in Step 3 above. You should see all of the directory's contents listed.



When working on a Quire project in a text editor like Atom or Visual Studio Code you can see and access all your project files from the sidebar at the left.

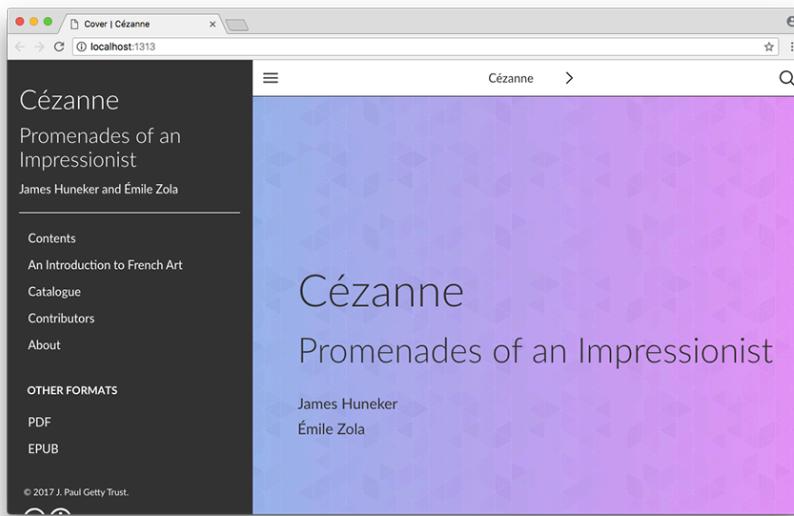
5. Entering Publication Metadata

The metadata for your publication (its title, subtitle, contributors, publication date, etc.) is used in various areas of your site. It's used under the hood for [search engine optimization](#) (SEO), as well as on the site itself in headings, navigation labels, and on your About or Copyright page.

All publication metadata is in the `publication.yml` file for your project, which you'll find in the `data` directory.

Open the `publication.yml` file and try changing the title and subtitle, saving the changes, and then looking at the preview running in your browser. You'll see that the cover is updated with your new title. It's also updated in the top navigation bar and in the expandable menu on the site. As much as possible, Quire works on the principle of having content exist only in one place in your files and using code to display it in multiple places in the publication as needed. This means when you make a change to something, you only do it once and it changes everywhere.

BETA



Basic publication information lives in the `publication.yml` file, and is used throughout the site. Here we see the title being used on the cover, in the top navigation bar and in the side navigation.

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

```
title: "Cézanne"  
subtitle: "Promenades of an Impressionist"
```

While not always necessary, it's usually a good idea to wrap any information you're entering in straight quotes as in the example above. Certain character combinations can otherwise cause issues with the way the YAML data is processed and may cause your site preview to fail.

Changes made to YAML files sometimes don't preview right away in the browser. If refreshing the browser doesn't work, stop and restart the `quire preview` process in your command-line shell.

For continuing issues, or if you just want to double check it, copy and paste your YAML text into an online YAML validator, which will alert you to any formatting errors.

The three other metadata files in the `data` directory—`figures.yml`, `references.yml`, and `objects.yml`—are data for figure images, bibliographic references, and catalogue/artwork objects. If your

publication won't have figures, a bibliography, or a catalogue section you can remove these. The `publication.yml` file, however, is always required and the more complete you can make it, the better.

Read more in the “Metadata & Configuration” chapter of this guide and our “Quire YAML” reference.

6. Editing Content

Next let's look at the `content` directory of your publication. In this directory are a series of **Markdown** files (`.md`) that hold the content of the publication. Each one represents a page of your website. The filename becomes part of the URL for that page in your final publication, so it's always lowercase and includes no spaces or special characters.

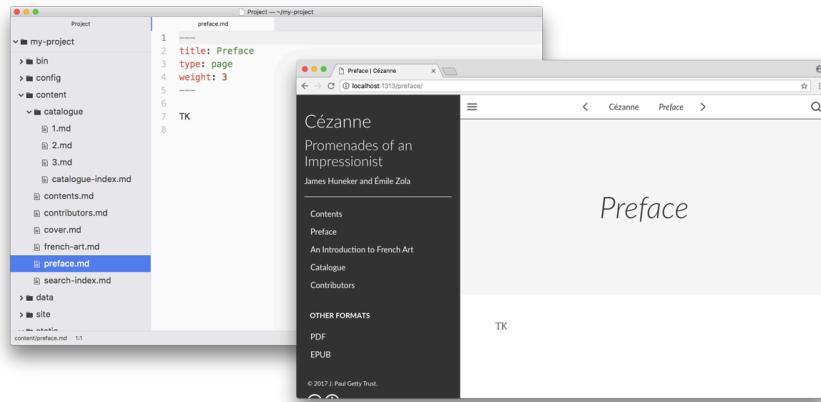
Open the `about.md` file. At the top you'll see a small block of YAML surrounded in two sets of three dashes `---`. Just like the overall publication has metadata, each page has metadata as well. Page metadata appears at the top of every Markdown file. Included in this example are the three most basic types of page YAML that you'll want to always include: the `title`, `type` and `weight`.

Let's make some changes to the `about.md` file to make it the Preface in our demo book:

1. First change the `title` to `"Preface"`.
2. The `type` of `"page"` is fine to leave as is. Other page types available are `"essay"`, `"entry"`, `"cover"`, and `"contents"`. Each displays the page content and data differently. The default is `"page"`.
3. The `weight` value creates the ordering of pages in your book. Without a `weight` value, Quire will automatically put the pages in order based on their filenames. Change the `weight` of this page to `"3"`, which will order it after the `contents.md` page which has a `weight` of `"2"`.

Save your changes and you should see the page update at `http://localhost:1313/about/`. You'll probably also want to change the name of the file from `about.md` to `preface.md`. Do this by right clicking (or control clicking on a Mac) on the file in your text editor and selecting “Rename”. Note that this will also change the URL of the page so instead of `http://localhost:1313/about/`, you'd now find the preview at `http://localhost:1313/preface/`.

BETA



Changing the page `title`, `weight` and filename of a Markdown file in your text editor (left), will change the page title, its ordering in the book, and its URL in the website (right).

Page content goes below the YAML block. Type or copy-and-paste some text here as a test. Save the file and check the preview you're running in the browser at <http://localhost:1313/preview/>. You should see the update.

Quire content is written in Markdown. Markdown allows you to express content structure as minimally as possible, using simple text indicators. For longer texts and publications, you'll want to use a Microsoft Word to Markdown conversion process, but you can also write Markdown directly in the text editor.

Use Markdown to add styles and elements to your sample text:

1. Add some italics to a phrase by surrounding it with asterisks: `*a phrase in italics*`.
2. Add a second-level heading by putting the text on it's own line, preceded by two hashmarks: `## Heading 2`.
3. Add a link by putting the link text in square brackets followed by the URL in parentheses: `[click here] (http://www.myurl.com)`.

```
## A Riotous Energy

There are many canvases the subjects of which are more
pathologic than artistic, subjects only fit for the
confessional or the privacy of the clinic. But, apart
from these disagreeable episodes, the main note of the
[Salon] (https://en.wikipedia.org/wiki/Salon\_\(Paris\))
is a riotous energy, the noisy ebullition of a gang
of students let loose in the halls of art.
```

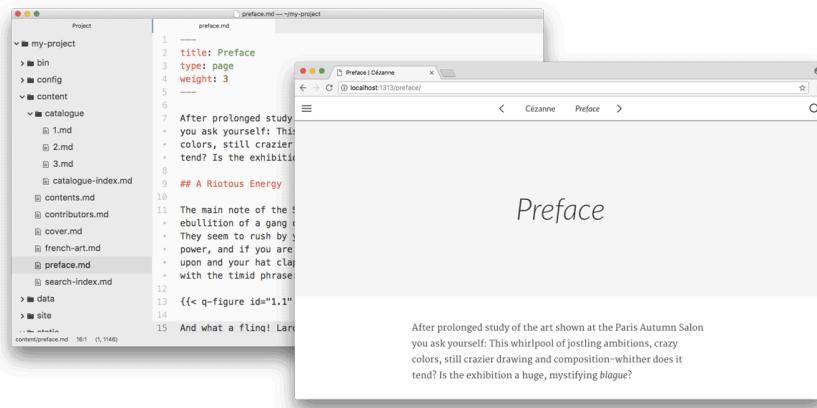
BETA



For more specialized features (especially images, multimedia, and citations), Quire extends Markdown's capabilities with a set of [shortcodes](#). Type the following figure image shortcode on a new line in your `about.md` file, save the change, and check the preview in your browser.

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

You'll see this added a figure and caption, the text for which is stored in the project's `figures.yml` file under the `id` of "1.1". If you update the information stored in `figures.yml` it will update on your page as well as anywhere else that figure is used.



The main text of a Quire publication is written in Markdown. Simple text formatting indicates structure. Like asterisks for italics (`*italics*`) and hashmarks for headings (`## Heading`). Figures are added with a “shortcode” that references a listing in your `figures.yml` file: `{{< q-figure id="1.1"`
`>}} }`

Read more about Page YAML, Markdown, and Quire shortcodes in the “Pages” chapter of this guide, and more about figure shortcodes in “Figure Images”.

7. Customizing Styles

There are number of different ways to customize the look of your publication. Some of the easiest are to add your own background images to your cover and page banners, and to change the colors and other styles of different interface elements (like the menu, navigation bar, and links)

BETA

with [CSS](#) variables.

A background image can be added to most pages by indicating the image in the page YAML of that page. Open the `cover.md` file, add the following line inside the page YAML, save the file, and preview the results in the browser.

```
image: cover_bg.jpg
```

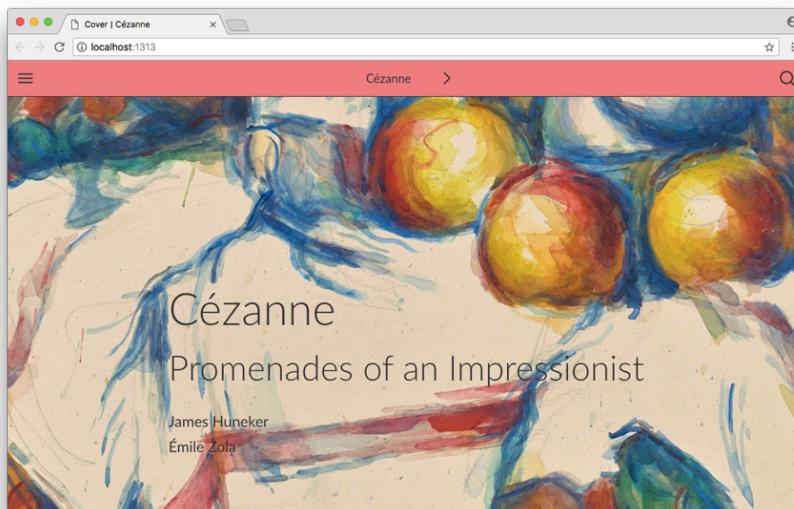
The image itself is stored in the `static/img` directory of your project, put any image file you'd like in there to use it as background image. You'll see that this is also where images are stored for use with the `q-figure` shortcode demonstrated above.

You can also change the colors used in various elements of the site design. Look inside the `themes` directory for the `quire-starter-theme/source/css` subdirectory and inside that, the `variables.scss` file. Here you'll find a number of variables, prefixed with a dollar sign, that are descriptive of what they control. For instance `$quire-navbar-color` is the background color of the navigation bar at the top of every page. Change it to something new—like a hex color value, or one of the standard 140 color names—save the change, and preview it in your browser. Like with changes to YAML files, changes to theme styles may require you to refresh the browser, or even to stop and restart the `quire preview` process in your command-line shell.

```
$quire-navbar-color: lightcoral;
```

Make sure there's always a space between the colon and the value you enter, and that the value is immediately proceeded by a semicolon. And as in the example above, your color choice won't be preceded by a dollar sign, even though some of them in `variables.scss` are. Hex color values are preceded by a hashmark (like `#ff00ff`) and the 140 standard color keywords don't need anything.

BETA



A background image can be added to the cover by adding it to the page YAML of your `cover.md` page (`images: cover_bg.jpg`). Colors and other styles can be customized in the `variables.scss` file of your project's theme.

Read more about applying your own custom CSS styles, altering page templates, and creating a new theme in the “Customizing Styles” chapter of this guide.

8. Outputting Your Publication

In your Terminal, stop the `quire preview` process by typing Control-C. To create the PDF version of your publication type `quire pdf` and press enter. For the EPUB, type `quire epub` and press enter. Both files will be created and saved into your project's `static/downloads` directory. View them by right clicking (or control clicking on a Mac) on the file name in the lefthand sidebar of your text editor and selecting “Show in Finder” or “Show in File Explorer”.



You can run the output commands anytime, and re-run them to update as you make changes to your project files.

For the online edition, type `quire site` and press enter. A `site` directory will be created in your project, with all the website files inside. The files in `site` can then be copied onto virtually any web server or hosting service. They include all the page content, images, and styles for the site and are all you need for the site to look and run just as it does when running `quire preview` on your own computer.

Read more about outputting your publication files and deploying your site in the “Multiformat Output” chapter of this guide.

Congratulations! Now What?

Congratulations on completing the tutorial! We've touched on Quire's core concepts and functionality, but of course there's more to learn and do.

- Watch a 90-minute webinar recording that walks through many of the steps in this tutorial.
- Sign up for beta access, if you haven't already, to work with Quire on your own.
- Continue reading the docs.
- Start working on your own Quire project.

BETA

Install or Update

[macOS Installation](#)

[Windows Installation](#)

[Linux Installation](#)

[Updating the Quire CLI](#)

macOS Installation

Open your Terminal [command-line shell](#) (found in your Applications/Utilities folder) and follow the steps below. First to install support software for Quire, and then Quire itself. If you are new to the command line, read our tutorial on “Working in a Command-line Shell”.

1. Install [Apple’s Xcode](#) by copying and pasting the following command and pressing enter. If Xcode is not already installed, an additional alert notification will pop up. Click “Install” and follow the prompts.

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

2. Visit the [Node.js](#) site, and download and install the current LTS (long-term support) version: <https://nodejs.org>.
3. Visit the [Pandoc](#) repository on GitHub, and

[Updating the Theme](#)

[Installing a New Theme](#)

[Uninstalling Quire](#)

[Troubleshooting](#)

download the macOS .pkg file of the latest release: <https://github.com/jgm/pandoc/releases/>. Install by double clicking the icon and following the prompts.

4. Visit the [PrinceXML](#) site, download the Mac OS version, and uncompress the folder: <http://www.princexml.com/download/>.

Copy and paste the following two lines in your Terminal at once and press enter. Note that this assumes the file downloaded into your Downloads folder (the Mac default) and that the version you downloaded was 12.2, you can update the command if either of these is not the case.

```
cd Downloads/prince-12.2-macosx
sudo ./install.sh
```

The Terminal will ask for your computer password. After entering it another

message will appear in the Terminal shell to confirm that PrinceXML should be installed in the `/usr/local` directory. Press enter.

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

```
cd
```

5. Copy and paste the following line into your Terminal to download the **Quire CLI (command-line interface)** to your computer from GitHub.

```
git clone https://github.com/gettypubs/
```

You may be asked to enter your GitHub username and password. When complete, install the CLI by copying and pasting the following commands into your Terminal and pressing enter. The first resets user permissions for your local directory, the second installs Quire.

```
sudo chown -R $USER /usr/local
```

```
cd quire-cli  
npm install -g
```

When complete, type `quire --version` to confirm proper installation. It should return a version number, otherwise, if it says `command not found`, the Quire CLI has not been properly installed. Refer to the “Troubleshooting” section below.

Windows Installation

1. Download **Git for Windows** by clicking on “Download” at <https://gitforwindows.org/>. An exe file will be downloaded, 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. Download and install **Node.js** and **npm** at <https://nodejs.org/en/download/>. Make sure you get the LTS version of Node.js, npm will be installed during the same process. The Windows installer will be downloaded, just open it and a setup wizard screen will guide you through the process.
3. Download **Prince** for Windows. You would download either the 32-bit installer or the 64-bit installer depending on your operating system. To install Prince you can follow the instructions at their site: <https://www.princexml.com/doc-install/#windows>
4. Download **Pandoc** for Windows. You would download either the 32-bit installer or the 64-bit installer depending on your operating system. To install Pandoc you can either download the .msi or download the .zip file and run the .exe file in the directory. Either way works and will install Pandoc on your system.
5. Open **PowerShell** (that should be installed by default in your Windows computer) and run it as administrator. To do so, right click on the icon and select “run as administrator” from the context menu.

Then once you get the administrator PowerShell window, type the following command:

```
npm install --g --production windows-bu
```

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 prompt with your computer name and your username once it’s complete.

6. Continue using PowerShell, but this time it’s not required to run it as administrator (you can close the administrator window and open a new one) and type the following

command to download Quire CLI to your computer from GitHub:

```
git clone https://github.com/gettypubs/
```

A pop up window may emerge and ask you to type your GitHub username and password, then the download starts.

Change directory to the `quire-cli` folder:

```
cd quire-cli
```

and type:

```
git checkout pc-dev
```

The following command will install dependencies:

```
npm install -g
```

Installing the dependencies takes some time and you'll notice multiple text strings being generated on the screen during the process. Wait until the prompt with your username shows up and then, to verify that `quire-cli` has been installed correctly, type:

```
quire -v
```

If version number is returned, means that the install is correct.

- Lastly, to navigate to your home directory and create a new project or publication type:

```
cd ~
```

Linux Installation

- Open terminal
- Download and install Prince:

```
cd ~
```

Run these commands to download and install PrinceXML

```
wget https://www.princexml.com/download/pri
```

```
sudo gdebi prince_10r2-1_debian8.0_amd64.de
```

If you chose a different distribution of Linux there are more instructions here -> <https://www.princexml.com/doc-install/#linux>

- Install node js and npm through nvm

Detailed information about this installation can be found in this link <https://github.com/creationix/nvm#install-script>

To install or update nvm, you can use the install script using cURL:

```
curl -o- https://raw.githubusercontent.com/
```

or Wget:

```
wget -qO- https://raw.githubusercontent.c
```

The script clones the nvm repository to `~/.nvm` and adds the source line to your profile (`~/.bash_profile`, `~/.zshrc`, `~/.profile`, or `~/.bashrc`).

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/
```

Note: You can add `--no-use` to the end of the above script (... `nvm.sh --no-use`) to postpone using `nvm` until you manually `use` it.

BETA

You can customize the install source, directory, profile, and version using the `NVM_SOURCE`, `NVM_DIR`, `PROFILE`, and `NODE_VERSION` variables. Eg: `curl ... | NVM_DIR="path/to/nvm"`. Ensure that the `NVM_DIR` does not contain a trailing slash.

NB. The installer can use `git`, `curl`, or `wget` to download `nvm`, whatever is available.

Note: On Linux, after running the install script, if you get `nvm: command not found` or see no feedback from your terminal after you type:

```
command -v nvm
```

simply close your current terminal, open a new terminal, and try verifying again.

If the above doesn't fix the problem, open your `.bash_profile` and add the following line of code:

```
source ~/.bashrc
```

For more information about this issue and possible workarounds, please refer here To verify that nvm has been installed, do:

```
command -v nvm
```

which should output 'nvm' if the installation was successful. Please note that `which nvm` will not work, since `nvm` is a sourced shell function, not an executable binary. To download, compile, and install the latest release of node, do this:

```
nvm install --lts
```

And then in any new shell just use the installed version:

```
nvm use --lts
```

1. Install Pandoc for EPUB and MOBI output

```
cd ~
```

Visit <https://github.com/jgm/pandoc/releases> and download the latest .deb file

```
wget https://github.com/jgm/pandoc/releas
```

Install the .deb file

```
sudo dpkg -i pandoc-2.2.3.2-1-amd64.deb
```

This will install the pandoc and pandoc-citeproc executables and man pages.

Verify pandoc was installed

```
pandoc -v
```

If Pandoc version information is returned then Pandoc was successfully installed on your system.

1. Install Quire-CLI

```
git clone https://github.com/gettypubs/quir
```

```
cd quire-cli
```

```
git checkout pc-dev
```

Install Dependencies

```
npm install -g
```

Verify

```
quire -v
```

If version number is returned quire-cli was installed correctly, you can now leave the directory.

```
cd ~
```

Updating the Quire CLI

As we develop, you may also want/need to update your Quire CLI. The CLI is pegged to a particular version of the Quire Starter Theme (at least for now), so if you're using an older CLI, any new projects you start will have the corresponding older version of the theme.

1. In your User/Home directory where you initially installed Quire, look for the `quire-cli` folder and delete it.
2. In your command line shell, enter the following three commands in order:

```
git clone https://github.com/gettypubs/
```

```
cd quire-cli
```

```
npm install -g
```

You now have the latest Quire CLI and any new projects you start will also have the newest theme. Your previously-started projects will keep their original version of theme unless you update the theme separately.

You may in some cases see errors or issues when running Quire commands with a newer version of the CLI, in older projects. These can be fixed manually, or, you can also re-install your original version of the CLI to run those

older projects if necessary.

Updating the Theme

To update the version of a theme you have:

1. In the `themes` directory of your project, delete the current theme directory. It's `quire-starter-theme` by default.
2. Copy in your new theme directory ensuring that it has the same name as the original.
3. 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 main user directory, you'd enter `cd my-project`.
4. 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 adding a new or replacement theme.)



Be sure to save any customizations you've made inside your theme. (Typically style changes to the `variables.scss` file.) You'll have to copy these over into the new theme manually once it is installed.

Installing a New Theme

TK

Uninstalling Quire

To uninstall Quire:

1. From a new window in your command-line shell (you should be in your home/user directory where you initially installed

BETA

Quire) type the following two commands:

```
cd quire-cli
```

```
npm uninstall -g
```

2. In the folder/finder view, go to your user/home directory where you initially installed Quire, look for the `quire-cli` folder and delete it.

Troubleshooting

TK

Getting Started

[Starting a New Project](#)

[Copying an Existing Project](#)

[Which Quire Files Are for Content Creators and Editors?](#)

Starting a New Project

To create a new project, 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
```

The process may take a minute as Quire installs the starter kit, configures the project, sets up the theme, and installs the dependencies. If you are using the Beta, you may need to enter your GitHub username and password twice during the download process: once for the starter kit and again for the starter theme.

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

[Which Quire Files Are for Developers?](#)

[Creating a Publication Outline](#)

[Prepping Images and Text](#)

[Previewing and Editing a Project](#)

Copying an Existing Project

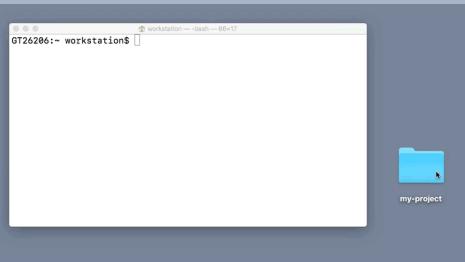
In addition to starting a Quire project from scratch as described in the previous section, you can also copy and work on a pre-existing Quire project. You would do this if you were on a team working on a publication together and are sharing the files via GitHub or another service, or you wanted to use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

1. Copy the Quire project directory into your main home/user 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 main user directory, you'd enter `cd my-project`.

- 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.)



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.



directory is 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*.



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 templates **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 (*Outputting & Deploying Your Site*); 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

Which Quire Files Are 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 publi
? data         <-- YAML files with publicati
? README.md
? site
? static       <-- Images / Style overrides
? themes        files that are output wit
```

? content

The central part of a Quire, the `content`

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. Read more in *Outputting & Deploying Your Site*.

Which Quire Files Are 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`, and `site` and `theme` directories.

```
? bin          <-- Scripts
? config       <-- Secondary/environmental c
? config.yml   <-- Main configuration
? content
? data
? README.md
? site         <-- The built site output wit
? static
? themes       <-- Layouts, shortcodes, styl
```

? bin

Currently it only contains a `deploy.sh` script file for deploying a Quire project to GitHub pages. Read more in *Outputting & Deploying Your Site*.

? config.yml

This is a standard, required file for [Hugo](#), and so 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

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 *Outputting & Deploying Your Site*.

? site

Where the built pages of the Quire website will be. This folder and its contents are automatically generated with the `quire site` command in the Quire CLI, and should not be edited directly. Read more in *Outputting & Deploying Your Site*.

? 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 in the Quire CLI, the default theme is `quire-starter-theme`. Read more in *Customizing Styles*.

Creating a Publication Outline

It is a good idea to start any project by creating a basic outline of your publication. 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 ordering, organization, and file naming 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 <code>cover.md</code> file	<code>mypublication.com/cover/</code>
The <code>contents.md</code> file	<code>mypublication.com/contents/</code>
The <code>chapter-01.md</code> file inside the <code>part-one</code> directory	<code>mypublication.com/part-one/chapter-01/</code>
The <code>section-overview.md</code> file inside the <code>part-two</code> directory	<code>mypublication.com/part-two/section-overview/</code>



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. **Filenames should be lowercase, with no spaces or special characters.**
2. **Sub-directories can't have other sub-directories within them.** Quire currently supports only one level of nesting.
3. **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.

Prepping Images and Text

TK

Previewing and Editing 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 main user 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.

BETA

Quire files can be edited in any text editor, though we recommended Atom or Visual Studio Code, two free and fully featured options. With one of these text editors installed, open your Quire project in it. You will see the directory contents listed in the left sidebar. As you make and save changes to these files the web browser preview of the site will automatically update as well.



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 restart the `quire preview` command in these cases.

Fundamentals: YAML & Markdown

[YAML Basics](#)

[Markdown Basics](#)

[Heading 2](#)

[Heading 3](#)

[Markdown Formatting Gotchas](#)

[Markdown Footnotes](#)

Content is stored in two different plain-text formats in Quire: YAML 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, the configuration files, and at the top of every Markdown file.

YAML Basics

Dashes represent individual items in a list. In the example below, there are two contributors, each with a first and last name. Note too, indentations matter in YAML.

```
item:
```

[Fractions, Superscripts and Subscripts](#)

[Markdown Preview](#)

[Markdown Output Configuration](#)

[Markdown Resources](#)

[Microsoft Word to Markdown Conversion](#)

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

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
```

Not all Quire YAML attributes expect Mark

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

In pages with the extension `.md`, YAML block entries can be in any order. It doesn't matter if you write:

```
---
```

```
title: Cheatsheet
```

```
type: page
```

```
---
```

Or:

```
---
```

```
type: page
```

```
title: Cheatsheet
```

```
---
```

You'll also notice that YAML values don't necessarily need to be in quotes. `title: My chapter`, without *My Chapter* in quotes, works just fine. However, 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 these cases, double quotes can minimize build issues:

```
title: "My Chapter".
```

```
item: "If the text here has a colon : or ot
```

Anything at all can go within double-quotes, except for other double-quotes. If you need double-quotes, use “curly quotes”, use a backslash to escape the double quote `\"` or Hugo will also process two single straight quotes “ as a

double when building the pages.

It's always a good idea to copy and paste your YAML blocks into a validator like YAML Lint or Code Beautify validator to make sure there aren't any hidden errors. You don't need to use the re-formatted YAML validators like this will sometimes give back to you, you just need to make sure it's correctly formatted.

Markdown Basics

Markdown is a very 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. By design, and through the stylesheets in the project theme, a list or ablockquote might look different from theme to theme, or even from format to format.

Here are the most commonly used tags:

```
*Italic Text*
```

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

Italic Text

Bold Text

```
# Heading 1
```

```
## Heading 2
```

```
### Heading 3
```

```
#### Heading 4
```

Heading 1

Heading 2

Heading 3

Heading 4

On the headings, in general you should not use the *Heading 1* tag in your content as that should be reserved for the page title, which is automatically generated. Start with *Heading 2*. Also headings should be thought of as levels of your content outline, not as sizes large to small, though they're often thought of and used that way. See our *Accessibility Principles* for more on this.

> Blockquote

Blockquote

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

[Link Text] (<http://www.linkaddress.com>)

Link Text

- dashes make
- a list with
- bullets

- dashes make
 - a list with
 - bullets

- * asterisks make
 - * a list with
 - * bullets

- asterisks make
 - a list with
 - bullets

Indented dashes and asterisks create sub bullets in a bulleted list.

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

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

Markdown Formatting Gotchas

- Individual paragraphs are created with double line breaks.
- Special characters like en- and em-dashes, and diacritics also 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.
- (c) will automatically render as ©.
- 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 `` 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't 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. Read more about Quire shortcodes below.

Markdown Footnotes

Footnotes can also be added with Markdown. Use a numbered marker like [^1] in the text where you want the number to go, and then at

the end of the page, usually under a “Notes” heading, add the corresponding footnote:

```
## Notes
```

[^1]: The footnote itself is the same thing

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 br

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

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.

[Blackfriday](#), the built in Markdown processor, will incorrectly also create link 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, (, for the first parentheses, or a backslash escape character before the first parentheses:

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

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, ⅓, ⅔, ⅚. 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 × 24³₁₆ inches = 19 × 24³/₁₆ inches
- 20th Century Sculpture = 20th Century Sculpture
- Chrome yellow (PbCrO₄) = Chrome yellow (PbCrO₄)

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 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](https://github.com/russross/blackfriday) (<https://github.com/russross/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 to insert any images and media to the Word document before conversion.
- Headings must be formatted with Word styles instead of by changing font formats.

While there are a number of free tools, we recommend using [Pandoc](#). You can find installation instructions on their site.

Once installed, make sure you are in the folder where your .docx document is saved, and enter the Pandoc command on the command-line application. The command used to convert your file is: `pandoc -s your_document_name.docx -t markdown -o your_document_name.md`. However, in order to optimize the default conversion you should specify the following extensions:

- Quire needs [ATX-style Markdown headers](#), these are specified adding `--atx-headers` to the command.
- The lines of the generated [Markdown 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-headers --wrap=none -s
your_document_name.docx -t markdown -o
your_document_name.md
```

BETA

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

Metadata & Configuration

[Adjusting the Default Publication Settings in the `config.yml` File](#)

[Adding and Editing Important Metadata in the `publication.yml` File](#)

[Title & Description](#)

[Publication Details](#)

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](#) and check out a sample of the [publication.yml](#) file in other chapters of this guide.

[Contributors](#)

[Copyright & License](#)

[Formats, Resources & Links](#)

[Subjects](#)

[Revision History](#)

Adjusting the Default Publication Settings in the `config.yml` File

The `config.yml` file is a standard and required file for [Hugo](#), and so 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 those 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 `publication.yml` file inside the `data` directory for that purpose (see below).

The properties in the `config.yml` file are individually documented in the [API/Docs 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.

Adding and Editing Important Metadata in the `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 [API/Docs](#) 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. And 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 to consider 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-b498182ad6c

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

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

The editors, designers and 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 “© 2018 Author Name”.

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, De
```

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/  
- type: getty  
  name: "Eugène Delacroix"  
  identifier: http://vocab.getty.edu/ulan
```

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](#).

Pages

[Defining Page Types](#)

[Putting Pages in the Right Order](#)

[Creating Section Landing Pages](#)

[Displaying your Pages](#)

[Formatting Text Content with
Markdown](#)

Every page in a Quire publication starts with a block of YAML (discussed further in the *Metadata & Configuration* page of this guide). 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:
---
```

Defining Page Types

The page `type` must be one of nine possible

[Applying Types of Linking](#)

[External Links](#)

[Internal Links Between Pages](#)

[Internal Links Within a Page / Between
Elements](#)

[Using Shortcodes to Add Features](#)

values: `page`, `essay`, `entry`, `cover`, `contents`, `about`, `search`, or `data`. If left blank, or if any other value besides these nine is entered, the `type` will default to “page”.

`page`: The default value. A general publication page. Used for introductions, forewords, chapters, appendices and other pages.

`essay`: Meant to be a standalone, self-contained article in a periodical or collected volume. The `essay` page type will typically include the contributor names and potentially bios, its own bibliography, and an abstract. This is also reflected in the metadata embedded in the page as well, which will describe the page more specifically and with more page-specific information than a typical publication page, whose metadata will instead point to the publication as a whole.

`entry` : An entry in a catalogue of objects.

Typically includes a large, zoomable image of the object, with a table of object details and narrative text. You can learn more about entries in the *Catalogues* page of this guide

`cover` : The cover of your publications.

Typically a full-screen splash page with large title, with cover description and publication information below. Along with defining `type: cover`, you will usually want to also add `slug: .` so that the cover page URL is the base URL of your site (your homepage). You can find more details about how to use `slug` in Creating section landing pages below.

`contents` : Automatically creates a table of contents for your entire publication, or for a section of your publication when used inside a sub-directory. The `tocType` can be set to either `full` or `short` in the publication's `config.yml` file. The `full` option will typically list and link to all main pages, sections and section pages. The `short` option will list and link to main pages and sections only, not the pages within the section. The `contents` page type can also be customized with four classes: `brief`, `list`, `abstract`, or `grid`.

`about` : The about page will typically include the publication description, citation information, revision history, downloads and other formats, copyright and license information and more. This can be used also for a traditional copyright page for the PDF output format.

`data` : Description TK

`search` : Description TK

Putting Pages in the Right Order

In this 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 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.

Creating 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, and in the *Table of Contents* and 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.”

Displaying your Pages

By default, every page you create will be included in all formats of your publication (online, PDF/print, and e-book) and will automatically be listed in the publication’s menu and *Table of Contents*. However, this can be overridden by setting any of the following attributes under `display` in the page YAML to

```
false :
```

```
display:
online:
menu:
pdf_ebook:
toc:
```

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 `public/` directory, simply delete any unneeded ones. Read more about site deployment in the chapter on *Multi-Format Output*.

Formatting Text Content with Markdown

The main content of your page appears after the YAML block at the top, 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 asterisks wrapping text turns it italic. You can learn more about Markdown syntax in the *Markdown fundamentals* section of this guide.

Applying Types of Linking

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] (http://www.getty.edu/museum/)
```

Internal Links Between Pages

Internal links between pages in your Quire publication can be included through the following Markdown formatting:

```
[Link text] (.../nameofpage/)
[Pea Pickers] (.../catalogue/2)
More info in our [about] (.../about/) page.
```

Internal Links Within a Page / Between Elements

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

- Callouts to figures (# are used for 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* section of this guide.

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

- Linking to other page elements (# is also used for other elements on the same page):
 - The IDs for other 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).
```

- Linking 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.

```
[referencetolink] (.../nameofpage/#idnamehere)
See the introduction [notes] (.../introduction).
```

Blackfriday, the built in Markdown processor, will incorrectly create link even if there is a space between the bracketed text and the parentheses when there is some text in brackets followed immediately by more text in parentheses. To avoid the linking, you can use a backslash escape character before the first parentheses.

- Footnotes within a page, NOT bibliography or citations shortcodes:
 - With this, a numbered footnote in the text when clicked upon will take a user to the corresponding area of the page where the full citation is located. The reverse is true from the linked number or arrow from the full citation.
 - This requires two parts:
 - Within the main body text, in a spot where you would like a footnote to be included will be set up as the following:
 - Example text. [^#]
 - "A photographic study of use of

- ```
leisure time in various income
groups."[^1]
```
- At the end of the main body text, one can designate a “Notes” or related subheading using ## or ###. Underneath that heading the information of the citation will be listed using this format:
  - [^#]: Corresponding citation information.
  - [^1]: Roy Stryker to all FSA (then RA) photographers, outline for first published in Carver, \*Just Before the War\*, n.p.
- You can find additional information about formatting notes text with Markdown in the *Markdown fundamentals* section of this guide.
- Citation and Bibliography shortcode linking:
  - When the citation shortcode, {{< q-cite "author date" "page # if applicable" >}} is used in body of text, that corresponds to the short and full bibliographic information provided in the `references.yml`, an in-page bibliography will be generated. This linking is completed automatically.
  - When the short code 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.

## Using Shortcodes to Add Features

Quire adds a number of specialty shortcodes which extend the functionality and possibilities of plain text markup. 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 (`{{< 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, some shortcodes act as wrappers around other text and so appear in pairs, with the second of the pair having a slash `/` preceding the shortcode name, much like you’d find in HTML markup.

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

The following shortcodes are (or will be) 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`: 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 css 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-contributors`: Can be used to create a page of contributor biographies, a section

## BETA

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.
- `q-link-list`: Creates an unordered list of links. Makes use of the `link-list.html` partial in the site templates.

# Collection Catalogues

## Capturing Object Data

### Creating Object Pages

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.

## Capturing 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 called as needed in different pages of your publication. Here is a brief sample:

## Generating Object Lists/Grids

```
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 x 63 cm (18.9 in x 24
 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/
 location: Cleveland Museum of Art
 link: http://www.clevelandart.org/art/1
```

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

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

## Creating 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 any essay/entry text included in the page [Markdown file](#).



### Reading

|            |                                              |
|------------|----------------------------------------------|
| Artist     | Berthe Morisot                               |
| Year       | 1873                                         |
| Dimensions | 74.2 x 100.3 x 12 cm (29 ½ x 39 ½ x 4 ½ in.) |
| Medium     | Oil on fabric                                |
| Location   | Cleveland Museum of Art                      |

[View in Collection](#)

The fashionable woman seated in the foreground is the artist's sister, Edma. However, the painting is not a portrait. Morisot's principal concern was to render a figure in a natural, outdoor environment. Edma's white dress—the prime vehicle for Morisot's study of reflected light—is saturated with delicate lavender, blue, yellow, and rose tonalities. Deftly executed with quick brushstrokes, the painting resounds with a feeling of freshness, vibrancy and delicate charm. "Every day I pray that the Good Lord will make me like a child," Morisot wrote, "That is to say, that He will make me see nature and render it the way a child would, without preconceptions." Morisot, the great granddaughter of the 18th-century French painter Jean-Honoré Fragonard, selected this painting as one of her four works shown in the first Impressionist exhibition of 1874.

screenshot of catalogue entry page as rendered in the browser

If you add multiple figures of the object, these are displayed in a rotating carousel, 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.

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.

## Generating 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 sub-directory within the main `content` directory (Read more about it in the *Pages and Plain Text* page of this guide).

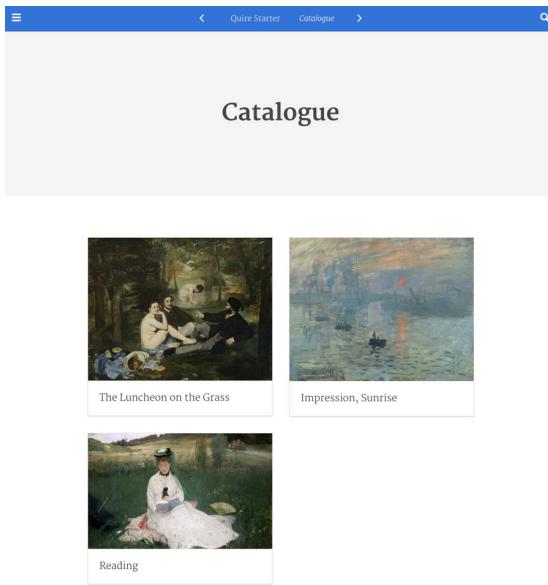
In this example, inside the `content` directory, we have a folder called `catalogue` and inside that, three numbered entries and an overview page:

```
? 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: .
```

## BETA



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.

screenshot of catalogue grid page as rendered in the browser

# Figure Images

[Including Figure Image Files in Your Publication](#)

[Creating a `figures.yml` File for Figure Image Metadata](#)

[Inserting Figure Images with the `q-figure` Shortcode](#)

[Labeling Figure Images](#)

Quire books are visual and the framework is built to support the use of images for scholarly purposes. In 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.

## Including Figure Image Files in Your Publication

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

[Styling Figure Images](#)

[Creating and Styling Figure Groups with the `q-figure-group` Shortcode](#)

[Adding Video Figures](#)

[Adding Basic Figures](#)

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.
- Only include images at as big a size as most readers will need. 800px on the longest side is fine for most figures, up to 1200px on the longest side for modest zooming. We find these sizes also work well enough in print.
- 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.

## Creating 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. These then can be called from wherever you need them 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_unitled96.jpg"
 alt: "detail of painting showing jagged b
- id: "1.2"
 src: "portrait-of-still.jpg"
 alt: "photograph of a frowning older man"
```

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



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.

## Images with the `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="" >} }
```



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" >} }`.

## Inserting Figure

# Labeling 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.

## Styling 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`, `is-pulled-right`, `is-large`, and `is-small`.

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 capriccio paintings, are typical in this respect as well as the image of storms and moonlight marines by Claude Joseph Vernet. So too a change from the rational and geometric French gardens of André Le Nôtre to the English gardens, which emphasized artificially wild and irrational nature. One

also finds in some of these gardens—curious ruins of temples—called “folies”.

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 Sévres

manufactures produced some of the finest porcelain of the time.

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



*Inspiration*, by Jean-Honoré Fragonard.  
Public domain image

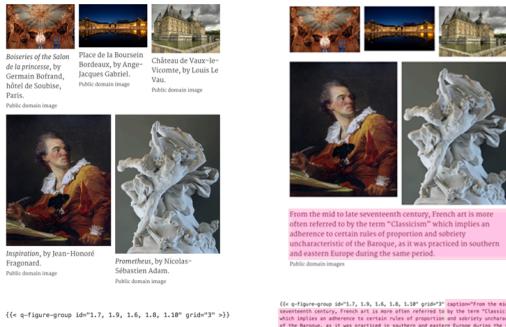
Some themes may offer additional options, and styles may be edited and new styles added in any theme with [CSS](#).

## Creating and Styling Figure Groups with the `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.

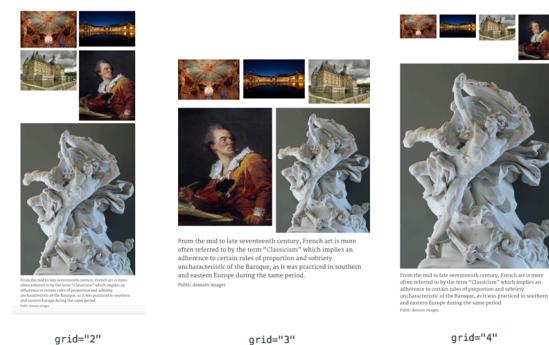


Just as with the single `q-figure` shortcode, classes can be added to groups to style them. For example, to create a small group of images running along one side of your text.

```
{< q-figure-group class="is-pulled-left" i
```

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
```



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.

## Adding 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
```

## Adding Basic Figures

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: "..".
```

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.

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
```

BETA

## *Zooming Images & Maps*

TK

# Citations and Bibliographies

## Capturing Bibliographic Information in YAML

### Adding Inline Text Citations

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.

## Capturing 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 `short` and a `full` form of the reference.

## Displaying a Bibliography

### Displaying the Short Reference in Bibliographies

```
entries:
 - short: "Faure 1909"
 full: "Faure, Élie. *Histoire de l'Art*"
 - short: "de Goncourt 1851"
 full: "de Goncourt, Edmond. *Journal de
```

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.

## Adding 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 can be output as a page-level bibliography on pages with a `type` of `"essay"` and `"entry"`, as explained below.

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

The shortcode can be used anywhere in your Markdown text, including within footnotes. The first positional parameter is a short form citation that should match one in `references.yml`. The second, optional parameter is a page reference (if needed). The above sample would output as: “Faure 1909, 54”.

The text element between the author date reference and the page (in this case, a comma and a space) can be changed with the `citationPageLocationDivider` property in your publication’s `config.yml` file according to your preferred citation style.

YAML block.

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

You may in some cases find that the system’s default sort method is sub-optimal. 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 such as `sort_as` 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"
```

## Displaying a Bibliography

Pages in your publication with a `type` of `“essay”` or `“entry”`, will automatically include a page-level bibliography listing all works that were cited on that page using the `q-cite` shortcode.

However, 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 >}}
```

You may create one page in your publication where the bibliography will be displayed. All you need is to add the `q-bibliography` shortcode after the

This custom key would need to be added to *all* entries, not just the one that need to be sorted differently than the default.

## Displaying the Short Reference in Bibliographies

Bibliographies displayed on pages with a `type` of `essay` or `entry`, and those generated with the `q-bibliography` shortcode, can be just 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”`.

Additionally, to change the sort, the `q-bibliography` shortcode accepts an optional `sort` value, which will sort the list by whatever key from the entries is given. Often

## BETA

"short" or "full".

BETA

## *Copright & About Pages*

TK

# Contributors

## Adding Contributors to Your Project

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.

## Adding Contributors to Your Project

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

Each `contributor` should have a `type` specified of either `"primary"`, `"secondary"`, or `"project-team"`.

- **Primary** contributors are authors, editors and others who would appear on a publication's cover or title page. In Quire **templates**, they are typically used on the cover, the menu and in the metadata embedded in your publication; listed as the creators of the publication.
- **Secondary** contributors are not typically

## Displaying a List of Contributors

shown in the menu or cover, but are often included in book metadata, and may also be referenced by the `q-contributor` shortcode as noted below. In future iterations of Quire, we hope to be able to reference contributors listed here from individual pages.

- **Project team** contributors are those that worked on the publication (editors, designers, developers, and the like), and can be listed using the `q-contributor` shortcode, often on an *About* or *Copyright* page. Project team contributors are not otherwise included in publication metadata, or used in other Quire **template**s.

If your publication is a monograph you might add the main authors to the `contributor` attribute in `publication.yml`, while if the publication has essays written by different authors, these would be added to the `contributor` fields on each essay/page YAML block.

Wherever they are listed (`publication.yml` or

pages YAML block), the following YAML attributes can be used for your contributors:

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

Not all of these are required. Depending on your usage, you may need as little as `first_name` and `last_name`, or just `full_name`. See the `q-contributor` shortcode reference for details on each of the standard contributor attributes.

Contributors may also be listed in the `contributor_as_it_appears` field in your `publication.yml` file. This value will override the individual `primary_contributor`s listed on the cover, the menu and in the book metadata. Useful when you want to include specific language about their role. For example: “Edited by Jane Smith and John Doe”.

## Displaying a List of Contributors

You can use the `q-contributor` shortcode 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 `“type”` value.

Sample: `{{< q-contributors range="page" type="bio" >}}`

The `“range”` value determines which contributors will be included in the list. Possible `“range”` values are:

| Value                     | Description                                                                                                                                                 |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>page</code>         | Only the contributors listed for the page the shortcode appears on.                                                                                         |
| <code>essays</code>       | Contributors on any page in your publications with <code>type: essay</code> .                                                                               |
| <code>all</code>          | All contributors.                                                                                                                                           |
| <code>primary</code>      | Contributors listed under <code>contributor</code> in your <code>publication.yml</code> file, and with a <code>type</code> of <code>“primary”</code> .      |
| <code>secondary</code>    | Contributors listed under <code>contributor</code> in your <code>publication.yml</code> file, and with a <code>type</code> of <code>“secondary”</code> .    |
| <code>project-team</code> | Contributors listed under <code>contributor</code> in your <code>publication.yml</code> file, and with a <code>type</code> of <code>“project-team”</code> . |

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

| Value                  | Description                                                                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>initials</code>  | Looks for the capital letters in a contributor first and last name and concatenates them together. Jane Pauley, becomes J.P.; Ralph Waldo Emerson becomes R.W.E. |
| <code>name</code>      | Just the first and last name.                                                                                                                                    |
| <code>name-plus</code> | The first and last name with, when available, their title and affiliation on a line below.                                                                       |
| <code>bio</code>       | First and last name, with pic, url, and bio as available. Plus a link to their contribution.                                                                     |

## BETA

There are some gotchas and limitations to the `q-contributor` shortcode:

- This shortcode can be used multiple times on a page, but ONLY if the same range is referenced.
- You can use a `file_as` value to control sort order. Otherwise lists are sorted alphabetically by `last_name`. 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.

- If a contributor is listed in many papers, the information in last listing will override all the others.
- Contributors with the same exact name will override each other. But using a `file_as` value would fix this (that value wouldn't show up in the interface). 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.

# Styles Customization

## Changing the Style Variables in the Theme

### Adding Custom Styles

### Overriding Theme Templates

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, or 4. creating an entirely new Quire theme.

## Changing the Style Variables in the Theme

Every Quire project has a **theme** inside the `themes` directory. When you first start a new project typing the `quire new` command in your command line interface, the default theme included is `quire-starter-theme`. In it, you can access simple text **Variables** that will let you update text and background colors, some element sizes, fonts, paragraph indents and more.

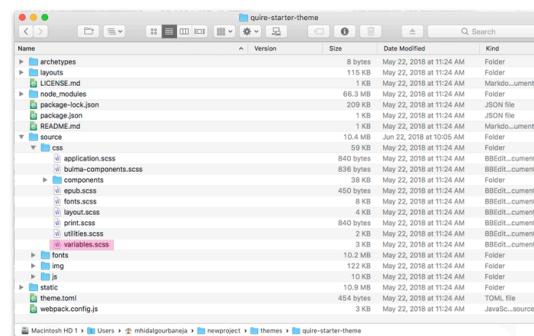
To find the variables, open the `quire-starter-theme` directory, navigate to the `source` sub-

## Creating a New Quire Theme

### Updating Your Theme to a Newer Version

### Changing to a New Theme

directory and then `css`, and open the file called `variables.scss`.



`variables.scss`

screenshot of the `variables.scss` file in the `quire-starter-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:

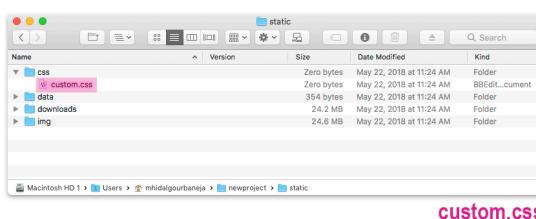
- The standard 140 [Color Keywords](#) such as `red`, `royalblue` and `honeydew`
- The many possible [HEX color values](#) like `#FF0000`, `#4169E1` and `#F0FFF0`
- [RGB Color Values](#) like `rgb(255, 0, 0)`, `rgb(65, 105, 225)` and `rgb(240, 255, 240)`

The `quire-starter-theme` uses the Bulma CSS framework, so you can also use any [Bulma variables](#).

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 sometimes need to refresh your browser page, or even clear the browser cache to get the style changes to fully load.

## Adding 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.

```
This text should be
```

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 specific page to be a different color than the titles all 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**).

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.

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.

## Overriding Theme Templates

**CSS** changes like those mentioned above are best for re-styling existing elements. If though 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 much 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 by 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="

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

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

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

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

If you copy the `essay` sub-directory 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="

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

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

<section id="content" class="section quir
 <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.

## Creating a New Quire Theme

TK

## Updating Your Theme to a Newer Version

Before updating your [theme](#), make note of any changes you made to it as these will need to be

manually copied over to the updated version of the theme if you want to keep them. Usually, this would only be changes to the style variables.

1. In the theme repository on GitHub, use the “Clone or download” button to download a ZIP file of the most current version of the theme, or, go to the repository’s Releases page to choose a particular release.
2. Once downloaded, unzip the package.
3. In the `themes` folder of your Quire project, delete the existing folder there, and replace it with the one you just downloaded. The replacement folder must be named the same as the original.
4. In your

#### SHORTCODE ERROR: q-glossary

The term supplied doesn’t match one in the project’s `glossary.yml` file. The term should be listed under the `term` or `alternates` key.

```
 {{< q-glossary "chiaroscuro" >}}
 term: "chiaroscuro"
 alternates:
 - "chiaroscuros"
```

, navigate to your project folder and run the command `quire install`. This will install your theme’s dependencies.

5. Once the install process is complete, manually copy over any changes/customizations to the theme that had made previously, and run `quire preview` to confirm.

You may also need to clear your browser cache to get the new theme stylesheets to reload.

## Changing to a New Theme

1. In the theme repository on GitHub, use the “Clone or download” button to download a ZIP file of the most current version of the theme, or, go to the repository’s Releases page to choose a particular release.
2. Once downloaded, unzip the package.
3. Add the new theme package to the `themes` folder of your Quire project. Leave the old theme there for now, until you confirm your new theme works and you are sure you want to use it.
4. In your project’s `config.yml` file change the name listed under `theme` from `"quire-starter-theme"` to the name of the new theme.
5. In your

#### SHORTCODE ERROR: q-glossary

The term supplied doesn’t match one in the project’s `glossary.yml` file. The term should be listed under the `term` or `alternates` key.

```
 {{< q-glossary "chiaroscuro" >}}
 term: "chiaroscuro"
 alternates:
 - "chiaroscuros"
```

, navigate to your project folder and run the command `quire install`. This will install your theme’s dependencies.

6. Once the install process is complete, run `quire preview` to confirm.

You may also need to clear your browser cache to get the new theme stylesheets to reload.

# Fonts Customization

## Customizing Fonts

### Adding a New Font

#### 1. Preparing Your Font Files and Adding Them to Your Project

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 `quire-starter-theme` to adding new external fonts.

## Customizing Fonts

The `quire-starter-theme` includes three embedded, open license fonts:

“Merriweather”, “Lato”, and “Aleo”. You can adjust which ones are used where in the “variables” file of your `theme`, `source/css/variables.scss`:

```
$serif: Merriweather, Georgia, serif;
$sans-serif: Lato, Helvetica, sans-serif;
$slab-serif: Aleo, Rockwell, "Trebuchet MS"

?family-primary: $serif;
// body text: $serif, $sans-serif, $slab-se
```

#### 2. Adding Font Information to Your Stylesheets

#### 3. Using Your New Font

```
$family-secondary: $sans-serif;
// headings and navigation items: $serif, $
```

The `$serif`, `$sans-serif` and `$slab-serif` variables tell your publication what fonts to use. For example, the variable `$serif: Merriweather, Georgia, serif;` tells Quire to use “Merriweather” as serif font. “Georgia” and a generic “serif”, the comma-separated fonts declared after our custom font, are fallbacks in case the browser doesn’t load or support our custom one (in this case “Merriweather”). You want your fallbacks to be of the same basic type as your custom one, and go from most to least specific in the list. Read more about fallbacks and `font-family` usage on Mozilla’s web docs.

The variables `$family-primary` and `$family-secondary` tell your publication where to use the fonts you specify with the above variables.

If the `$family-primary` font of your publication is `$serif`, “Merriweather” (and its fallback options) will be used in the body text of the publication pages.

## Adding a New Font

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 the 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. Preparing Your Font Files and Adding 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/quire-starter-theme` folder, all fonts are stored in `source/fonts`. Move your folder of fonts there.

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.

```
cooper-hewitt
```

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, if it’s of a concern, 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. Adding 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-light.woff");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-light.woff?#iefix");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-medium.woff");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-medium.woff?#iefix");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-normal.woff");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-normal.woff?#iefix");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-bold.woff");
 src: url("../fonts/cooper-hewitt/cooper-hewitt-bold.woff?#iefix");
 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.

<code>font-weight</code>	<code>Font name</code>
100	Extra Light or Ultra Light

<code>font-weight</code>	<code>Font name</code>
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. Using 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 Customizing 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:

## BETA

```
$family-primary: "Cooper Hewitt", Helvetica
```

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

# Multiformat Output

Text to come.

## Licensing

If the source files of your publication are hosted publicly on a site like GitHub, you should also include a `LICENSE.md` file to specifically define the terms of use for your files. A `LICENSE.md`

file is not included by default in Quire projects, but we recommend looking at other Quire projects for samples. Typically the file will include license information for both the content and text content of your publication (Creative Commons), and also for the code (GNU Public, MIT, or others).

# YAML

[Configuration](#)

[Black Friday Markdown](#)

[Quire Parameters](#)

[Publication](#)

[Publisher](#)

[Description](#)

[Subject](#)

[License](#)

[Resource Link](#)

[Revision History](#)

[Identifier](#)

[Contributor](#)

[Figure](#)

[Bibliography](#)

[Object](#)

[Page](#)

## Configuration

Location: `config.yml`

Type: Object

Object Properties	Expected Value	Description	Object Properties	Expected Value	Description
<code>baseURL</code>	url	The base url for your project.	<code>canonifyURLs</code>	boolean	Converts all internal links to being in complete canonical format. Default is
<code>blackfriday</code>	object	Options for Blackfriday, Hugo's			markdown renderer. See below.

Object Properties	Expected Value	Description	Object Properties	Expected Value	Description
		<code>false</code> .	<code>fractions</code>	boolean	When set to <code>true</code> any numbers separated by a slash are automatically converted to fractions. Default is <code>false</code> . Though even then $\frac{1}{4}$ , $\frac{1}{2}$ and $\frac{3}{4}$ are still converted. Recommend always using proper unicode fractions: $\frac{1}{4}$ , $\frac{1}{2}$ , $\frac{3}{4}$ , $\frac{1}{8}$ , $\frac{3}{8}$ , $\frac{5}{8}$ , $\frac{7}{8}$ .
<code>footnoteReturnLinkContents</code>	string	Controls the appearance of the link added to the end of footnotes. Default is “ <code>↔</code> ”.			
<code>metaDataFormat</code>	“yaml”, “toml”, “json”	Default is “yaml”			
<code>params</code>	object	Additional parameters for Quire. See below.			
<code>relativeURLs</code>	boolean	Keeps all internal links relative. Default is <code>true</code> .			
<code>theme</code>	url/id	The name of the theme, in the <code>theme</code> directory you’re using. Quire starter kit default is <code>quire-base-theme</code>	<code>displayBiblioShort</code>	boolean	When <code>true</code> the short citation form (ie., “Smith 2003”) will be displayed with the full form, when creating a bibliography on pages with <code>type: essay</code> , or on other pages with the <code>qbibliography</code> shortcode. Default is <code>true</code> .
			<code>citationPageLocationDivider</code>	string	Default is “,”.

See: Additional Hugo configuration options

## Black Friday Markdown

Location: `blackfriday` in `config.yml`

Type: Object

# BETA

Object Properties	Expected Value	Description	Object Properties	Expected Value	Description
<code>imageDir</code>	string	The project's main image directory inside the <code>static</code> directory. Default is <code>"/img/"</code> .	<code>short_title</code>	string	A short version of your title, primarily for use in navigation elements with limited space.
<code>figureLabels</code>	boolean	Default is true.	<code>url</code>	url	The full URL of your final publication.
<code>figureLabelsTextBefore</code>	string	Default is "Figure "	<code>pub_type</code>	"book", "journal-periodical", "other"	Can be one of three values. Determines how key search-engine metadata is defined.
<code>figureLabelsTextAfter</code>	string	Default is ". "	<code>pub-date</code>	YYYY-MM-DD	The first date your publication will be released
<code>videoFigureFallbackText</code>	boolean	Default is true.	<code>language</code>	2-letter ISO 639-1 language code(s)	Taken from the the list at <a href="https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes">https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes</a> . List multiple languages using a comma-separated list.
<code>videoFigureFallbackTextBefore</code>	string	Default is "Watch the video at "			
<code>videoFigureFallbackTextAfter</code>	string	Default is "."			
<code>menuType</code>	"short", "full"	Default is "short"	<code>identifier</code>	object	See below.
<code>searchEnabled</code>	boolean	Default is true	<code>publisher</code>	array	See below.
<code>tocType</code>	"short", "full"	Default is "full"	<code>series_periodical_name</code>	string	
			<code>series_issue_number</code>	string	
			<code>contributor</code>	array	See below.
			<code>contributor_as_it_appears</code>	string	
			<code>promo_image</code>	url	
<b>Object Properties</b>		<b>Expected Value</b>	<b>Description</b>	<b>Object Properties</b>	<b>Expected Value</b>
			<code>description</code>	object	See below.
			<code>subject</code>	array	See below.
<code>title</code>	string	The title of your publication.	<code>library_of_congress_cip_data</code>	list	
<code>subtitle</code>	string	The subtitle of your	<code>copyright</code>	string	

## Publication

Location: `publication.yml`

Type: Object

Object Properties	Expected Value	Description	Object Properties	Expected Value	Description
<code>title</code>	string	The title of your publication.	<code>library_of_congress_cip_data</code>	list	
<code>subtitle</code>	string	The subtitle of your	<code>copyright</code>	string	

# BETA

Object Properties	Expected Value	Description	Object Properties	Expected Value	Description
license	object	See below.			special features in one or the other of the formats.
resource_link	array	See below.	pdf_ebook	string	
revision_history	array	See below.			
repository_url	url	A public repository for source code and revision history for the publication.	subject		Location: subject in publication.yml
					Type: Array

## Publisher

Location: publisher in publication.yml

Type: Array

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

## Description

Location: description in 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

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

## License

Location: license in 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: “CCo”, “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.
icon	url	
scope	“text-	

# BETA

Object Properties	Expected Value	Description	Item Attributes	Expected Value	Description
	only”, “full”, “some-exceptions”				relations/link-relations.xhtml
online_text	string	Markdown okay. Will override the automatically generated license text for the online edition only.	url	url	URL to web resource or to download.
pdf_ebook_text	string	Markdown okay. Will override the automatically generated license text for the PDF and e-book editions only.	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	

## Resource Link

Location: `resource_link` in `publication.yml`

Type: Array

Item Attributes	Expected Value	Description
name	string	How the link will be named.
type	“other-format”, “related-resource”, “footer-link”	
media_type	string	Taken from the list at <a href="https://www.iana.org/assignments/media-types/media-types.xhtml">https://www.iana.org/assignments/media-types/media-types.xhtml</a>
link_relation	string	Taken from the list at <a href="http://www.iana.org/assignments/link-relation">http://www.iana.org/assignments/link-relation</a>

## Revision History

Location: `revision_history` in `publication.yml`

Type: Array

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

## Identifier

Location: `identifier` in top level of `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 <code>pub-type</code> of “book”. ISBNs can be purchased individually or in packages at <a href="http://www.isbn.org/">http://www.isbn.org/</a> .
issn	8-digit	For use with <code>pub-type</code> of

Item Attributes	Expected Value	Description
	ISSN	“journal-periodical”. ISSNs can be requested through <a href="http://www.issn.org/">http://www.issn.org/</a> .
	doi	string Not yet implemented.
	uuid	string Not yet implemented.

## Contributor

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

Type: Array

Item Attributes	Expected Value	Description
	<code>id</code>	
	<code>type</code>	“primary”, “secondary”, “project- team”
	<code>first_name</code>	
	<code>last_name</code>	
	<code>full_name</code>	
	<code>file_as</code>	
	<code>title</code>	
	<code>affiliation</code>	
	<code>role</code>	
	<code>role_code</code>	3-letter role code Taken from the list at <a href="https://www.loc.gov/marc/relators/relaterm.html">https://www.loc.gov/marc/relators/relaterm.html</a>
	<code>url</code>	
	<code>bio</code>	
	<code>pic</code>	

## Figure

Location: `figure_list` in `figures.yml`

Type: Array

Item Attributes	Expected Value	Description
	<code>id</code>	Numbers and lowercase letters only, with no spaces or special characters ( <code>001</code> , <code>fig-01a</code> , etc).
	<code>src</code>	Should be the file name of a JPG, PNG or GIF image ( <code>fig01.jpg</code> ). Avoid using spaces or special characters, and if it's in a sub-folder within the main <code>img</code> directory (which is defined by the <code>imageDir</code> parameter in the <code>config.yml</code> file), it should include that sub-folder name as well ( <code>comparatives/fig01.jpg</code> ).
	<code>alt</code>	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.
	<code>caption</code>	The caption to appear below the figure. Special characters are allowed. Use Markdown for formatting.
	<code>credit</code>	Follows the caption. Markdown allowed.
	<code>media_type</code>	Currently supports video hosted on YouTube or

# BETA

Item Attributes	Expected Value	Description	Item Attributes	Expected Value	Description
		Vimeo. (May eventually expand to HTML5 video, audio, and Soundcloud, and others.) When a <code>media_type</code> is defined, a <code>media_id</code> must be as well. For video, it is also recommended that an image <code>src</code> still be used (presumably being a screenshot from the video) so as to provide a fallback for PDF and EPUB output.			and <code>imageLabelContentAfter</code> values defined in your <code>config.yml</code> file.
				Type: Array	
Item Attributes	Expected Value	Description	Attribute	Expected Value	Description
<code>media_id</code>	string	The ID of the video resource on YouTube or Vimeo. For example, in the URLs <a href="https://www.youtube.com/watch?v=VYqDpNmnu8I">https://www.youtube.com/watch?v=VYqDpNmnu8I</a> or <a href="https://youtu.be/VYqDpNmnu8I">https://youtu.be/VYqDpNmnu8I</a> , the <code>media_id</code> would be <code>VYqDpNmnu8I</code> ; and in <a href="https://vimeo.com/221426899">https://vimeo.com/221426899</a> it is <code>221426899</code> .	<code>short</code>	string	The short form of the citation, ie., Brown 1984.
			<code>full</code>	string	The full form of the citation, ie.,
				Type: Array	
				Attribute	Description
				Value	
<code>aspect_ratio</code>	“standard”, “widescreen”	For use with video <code>media_type</code> s to properly scale video embeds. When no value is provided, the default is “widescreen”.	<code>id</code>	string	Required. Used to reference objects from entry pages. Should be numbers and lowercase letters only, with no spaces or special characters ( <code>001</code> , <code>fig-01a</code> , etc).
<code>label_text</code>	string	Used for the <code>q-figure-group</code> shortcode only. A short text label added to the image, usually just under the image depending on your theme. If no text is provided here, a label is automatically generated from the provided <code>id</code> value along with the <code>imageLabelContentBefore</code>	<code>figure</code>	array	A list of one or more images of the object. It is recommended that this list be only of <code>id</code> values corresponding with <code>id</code> s in your project’s <code>figures.yml</code> file.

## Bibliography

Location: `entries` in `references.yml`

Type: Array

Item Attributes	Expected Value	Description
--------------------	-------------------	-------------

<code>media_id</code>	string	The short form of the citation, ie., Brown 1984.
-----------------------	--------	--------------------------------------------------

<code>full</code>	string	The full form of the citation, ie.,
-------------------	--------	-------------------------------------

## Object

Location: `object_list` in `objects.yml`

Type: Array

Attribute	Expected Value	Description
-----------	-------------------	-------------

<code>id</code>	string	Required. Used to reference objects from entry pages. Should be numbers and lowercase letters only, with no spaces or special characters ( <code>001</code> , <code>fig-01a</code> , etc).
-----------------	--------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<code>figure</code>	array	A list of one or more images of the object. It is recommended that this list be only of <code>id</code> values corresponding with <code>id</code> s in your project’s <code>figures.yml</code> file.
---------------------	-------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# BETA

Attribute	Expected Value	Description
<code>link</code>	<code>url</code>	A URL link to a page with more/current information on the object. Usually the object in the museum's online collection pages.
<code>date_start</code> , <code>date_end</code>	<code>integer</code>	Reserved for future use in Quire.
<code>dimension_width</code> , <code>dimension_height</code> , <code>dimension_depth</code>	<code>integer</code>	Reserved for future use in Quire.
Objects also support arbitrary attributes, which might include <code>title</code> , <code>artist</code> , <code>collection</code> , etc. Those added will be output in a table on collection catalogue entry pages. The ordering of the display can be controlled with <code>object_display_order</code> in <code>objects.yml</code> .		
See: Guide on Collection Catalogues		
<b>Page</b>		
Location:	Any page in <code>content/</code>	
Type:	Object	
<code>number</code>	<code>integer</code>	# for chapter/paper numbering.
<code>title</code>	<code>string</code>	
<code>subtitle</code>	<code>string</code>	
<code>short_title</code>	<code>string</code>	
<code>type</code>	<code>"page", "essay", "entry", "cover", "contents", "about", "data", "search"</code>	
<code>class</code>		
<code>weight</code>	<code>integer</code>	
<code>object</code>	<code>array</code>	See above.
<code>contributor</code>	<code>array</code>	See above.
<code>abstract</code>	<code>string</code>	
<code>slug</code>		
<code>display</code>	<code>"online", "menu", "pdf_ebook", "toc"</code>	

The `object` and `contributor` attributes above are arrays of one or more items. The details of what YAML values each of those items can have, can be found in the Catalogue Entries and Contributors sections respectively.

# Shortcodes

[q-backmatter](#)

[q-bibliography](#)

[q-cite](#)

[q-contributor](#)

[type](#)

[range](#)

[q-copyright](#)

[q-figure](#)

## **q-backmatter**

Sample: {{< q-backmatter >}}The text you want formatted here.{{< /q-backmatter >}}

Basic Usage: Wrapping text in this shortcode will format it in a manner similar to how footnotes and bibliographies are formatted. Good for sections at the end of essays like Appendices, Acknowledgments, and Contributors Bios.

Parameters: None.

YAML Dependencies: None.

[q-figure-group](#)

[q-link-list](#)

[q-loc](#)

[q-publisher-team](#)

[q-revision-history](#)

[q-table](#)

[q-url-link](#)

See: Working with Text

## **q-bibliography**

Basic Usage: Generates a bibliography from the entries in the project's `bibliography.yml` file.

See: Citations & Bibliographies

## **q-cite**

Basic Usage: Adds a linked Author Date citation reference to the text, and a hover pop-up with

# BETA

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-contributor

Sample: `{< q-contributor range="page" type="bio" >}`

Basic Usage: 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.

Required Named Parameters: “range” and “type”

### type

#### Expected Description

#### Value

“initials”	Looks for the capital letters in a contributor first and last name and concatenates them together.
“name”	Just the first and last name.
“name-plus”	The first and last name with, when available, their title and affiliation on a line below.
“bio”	First and last name, with pic, url, and bio as available. Plus a link to their contribution.

### range

#### Expected YAML Location

#### Value

Expected Value	YAML Location	Description
“all”	.Site.Pages.Params.contributor	In all Pages under contributor
“primary”	.Site.Data.publication.contributor with a type of “primary”	In publication.yml under contributor
“secondary”	.Site.Data.publication.contributor with a type of “secondary”	In publication.yml under contributor
“project-team”	.Site.Data.publication.contributor with a type of “project-team”	In publication.yml under contributor

### YAML Structure:

#### YAML Attribute Required Notes

#### Attribute

Either both `first_name` Y If all are included, `full_name` will override the others.  
and  
`last_name`,  
OR

`full_name`

`title` N

`affiliation` N

`role` N When the shortcode range is “publication-team”, `role` is used in place of `title`

“page”	.Page.Params.contributor	In the current Page url bio contributor url	N	Markdown okay
“essays”	.Site.Pages.Params.contributor	In any Page with pic a type: essay	N	Filename only. File must be in the top level image

YAML Attribute	Required	Notes	Named Parameters	Expected Value	Description
		director, or in a sub-directory defined in <code>config.yml</code> with <code>contributorSubDir</code> .			in a shortcode <i>with</i> a corresponding <code>src</code> parameter, this will create an <code>id</code> for the image markup that can be used to link to the image directly ( <code>mypublication.com/chapter01/#fig-3</code> ) and ignores any potentially corresponding information in the <code>data/figures.yml</code> file.
See: Contributors					
<b>q-copyright</b>					
Sample: <code>{{&lt; q-copyright &gt;}}</code>		Basic Usage: Adds a copyright statement and licensing information as you've defined it in your <code>publication.yml</code> file. Commonly used on Copyright and About pages. The shortcode itself makes use of the <code>copyright.html</code> partial in your site templates.			
Parameters: None.					
See: Copyright & About Pages					
<b>q-figure</b>					
Sample: <code>{{&lt; q-figure id="3.1" &gt;}}</code>		Basic Usage: Inserts a formatted figure image, label, caption and credit line. If using a <code>data/figures.yml</code> file, only an <code>id</code> parameter is required for this shortcode. If other values supplied directly in the shortcode they will override any corresponding values in the <code>data/figures.yml</code> .			
Named Parameters	Expected Value	Description			
<code>id</code>	string	Spaces or special characters should not be used and will be stripped out. When used in a shortcode <i>without</i> a corresponding <code>src</code> parameter, the shortcode will look for a matching <code>id</code> in the project's <code>data/figures.yml</code> file. When used	<code>alt</code>	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.
			<code>caption</code>	string	The caption to appear below the figure. Special characters are allowed. Use Markdown for formatting.
			<code>credit</code>	string	Follows the caption. Markdown allowed.

# BETA

Named Parameters	Expected Value	Description	Named Parameters	Expected Value	Description
<code>label</code>	boolean	Default is set to <code>true</code> . <code>true</code> will add a label to the caption, such as “Figure 1.3”, <code>false</code> will remove the label. The global label setting is in the <code>config.yml</code> file under the parameter <code>figureLabels</code> .			corresponding values in the project’s <code>data/figures.yml</code> file.
<code>label_text</code>	string	Will override the default label text for the figure, which is otherwise constructed automatically with the <code>figureLabelsTextBefore</code> and <code>figureLabelsTextAfter</code> parameters in <code>config.yml</code> .	<code>caption</code>	string	The caption to appear below the figure group. Special characters are allowed. Use Markdown for formatting. Overrides any caption information provided in <code>data/figures.yml</code> .
<code>class</code>	<code>is-pulled-right</code> , <code>is-pulled-left</code> , <code>is-full-width</code> , <code>is-centered</code> , <code>small</code>	Sets the style of the figure image.	<code>credit</code>	string	Follows the caption. Markdown allowed. Overrides any caption information provided in <code>data/figures.yml</code> .
			<code>label</code>	boolean	Default is set to <code>true</code> . <code>true</code> will add a label to the caption, such as “Figure 1.3”, <code>false</code> will remove the label. The global label setting is in the <code>config.yml</code> file under the parameter <code>figureLabels</code> . If a <code>caption</code> 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.

See: Figure Images and Figure YAML

## q-figure-group

Sample: `{{< q-figure-group id="3.1, 3.2, 3.3" >}}`

Basic Usage: Like `q-figure`, but with handling for multiple images at once.

Named Parameters	Expected Value	Description	Named Parameters	Expected Value	Description
<code>id</code>	string	One or more comma-separated <code>id</code> s that match	<code>grid</code>	<code>1, 2</code>	Determines the horizontal

Named Parameters	Expected Value	Description
	3 , 4 , 5 , 6	width (in number of images) of the image grid. If no grid is set, the images will stack on top of one another vertically.

See: Figure Images and Figure YAML

## q-link-list

Sample: {{< q-link-list "other-formats" >}}

Basic Usage: Creates an unordered list of links. Makes use of the `link-list.html` partial in the site templates.

Parameters:

Parameter Position	Expected Value	Description
0 *	"other-formats", "related-resources"	Values point to corresponding values in <code>publication.yml</code>

Parameters are not named, but instead defined by their position, starting at 0.

See: Copyright & About Pages

## q-loc

Sample: {{< q-loc >}}

Basic Usage: Adds formatted Library of Congress Cataloging-in-Publication Data to the page, from values in `publication.yml`.

Parameters: None

See: Copyright & About Pages

## q-publisher-team

Note: this should be folded into the `q-contributor`.

## q-revision-history

Sample: {{< q-revision-history >}}

Basic Usage: Adds a revision history to the page, based on values in `publication.yml`.

Named Parameters	Expected Value	Description
format	"short", "full" "long"	"short" will show only the first publication date and most recently updated date. "long" shows all dates and list of changes for each.

See: Copyright & About Pages

## q-table

Note: Original version of this was tabled. Still looking for better solution for complex tables.

See: Figures

## q-url-link

Note: Not sure we'll do this. Might be better to build in with JS rather than have a separate shortcode just for URLs.

See: Working with Text

# Dependency Guide

[pe-epub and pe-epub-fs](#)

[Quire Dependency Tables](#)

[CLI Dependencies](#)

Each Quire project relies on two projects the `quire-cli` a command line interface (cli) to run commands to preview and build a static website, a PDF or an EPUB and the `quire-starter-theme` a front end development toolkit that allows users to shape the output of the website, PDF and EPUB. This page explains what makes these two projects work and what major dependencies currently make quire what it is.



*Hugo is a static HTML and CSS website generator written in Go. It is optimized for speed, ease of use, and configurability. Hugo takes a directory with content and templates and renders them into a full HTML website. Hugo relies on Markdown files with front matter for metadata, and you can run Hugo from any directory. This works well for shared hosts and*

[CLI Dev Dependencies](#)

[Starter Theme Dependencies](#)

[Starter Theme Dev Dependencies](#)

*other systems where you don't have a privileged account.*

Quire makes use of Hugo via the npm package `hugo-bin`

Quire uses Hugo's cli, templating system and http server. to create a way to preview your site while editing the front end code, build a static html site and aids Prince XML to create a PDF of your publication you are building.



*Prince can also be used by authors and publishers to typeset and print documents written in HTML, XHTML, or one of the many XML-based document formats. Prince is capable of formatting academic papers, journals, magazines, and books.*

Quire uses the output of Hugo static-site

generator to build a PDF as referenced above.



*At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.*

Quire makes use of Webpack via the npm package [webpack](#)

Currently Quire starter theme uses the latest version of Webpack 4 to bundle front end assets and support the development workflow.

To modify the Webpack configuration for your project edit this file

```
<your-project-directory>/themes/quire-
starter-theme/webpack.config.js
```

## pe-epub and pe-epub-fs

*“pee pub” makes epubs better. Our goal is to make it as easy as possible to output a valid epub. It’s used in production over at The People’s E-Book. pe-epub-fs extends pe-epub so you can import local assets from your filesystem rather than from the web.*

Quire uses these projects to generate the EPUB file. This file can be accessed on any device or software that reads the .epub file format. These projects generate a mostly style stripped version of the publication. These projects have been receiving limited maintenance. Quire is currently seeking a replacement to output the EPUB file.

## Quire Dependency

# Tables

## CLI Dependencies

Dependency	NPM Description
axios	Promise based HTTP client for the browser and node.js
chalk	Terminal string styling done right
cheerio	Fast, flexible & lean implementation of core jQuery designed specifically for the server.
command-exists	node module to check if a command-line command exists
commander	The complete solution for node.js command-line interfaces, inspired by Ruby’s commander.
execa	A better <code>child_process</code>
glob	Match files using the patterns the shell uses, like stars and stuff.
hugo-bin	Binary wrapper for Hugo
js-yaml	YAML 1.2 parser / writer for JavaScript
lodash	The Lodash library exported as Node.js modules.
[pe-epub]()	Makes epubs better.
[pe-epub-fs]()	Extends pe-epub so you can import local assets from your filesystem rather than from the web
rimraf	The UNIX command rm -rf for node.
striptags	An implementation of PHP’s strip_tags in Node.js.
webpack	webpack is a module bundler
yaml-front-matter	Parses yaml or json at the front of a string. Places the parsed content, plus the rest of the string content, into an object literal.

## CLI Dev Dependencies

Dependency	NPM Description
eslint	ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code.
eslint-config-standard	Shareable configs are designed to work with the extends feature of .eslintrc files. You can learn more about Shareable Configs on the official ESLint website.
eslint-plugin-promise	Enforce best practices for JavaScript promises.
eslint-plugin-standard	ESlint Rules for the Standard Linter
jsdoc	An API documentation generator for JavaScript.
jsdoc-template-argon	Template System for jsdoc
mocha	Simple, flexible, fun JavaScript test framework for Node.js & The Browser
tmp	A simple temporary file and directory creator for node.js.

## Starter Theme Dependencies

Dependency	NPM Description
bulma	Bulma is a CSS framework. There is no JavaScript included.
hammerjs	Hammer is a open-source library that can recognize gestures made by touch, mouse and pointerEvents
jquery	jQuery is a fast, small, and feature-rich JavaScript library.
leaflet	Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps.

Dependency	NPM Description
lodash	lodash is a fast, small, and feature-rich JavaScript library.
lunr	A bit like Solr, but much smaller and not as bright.
smoothstate	smoothState.js is a jQuery plugin that progressively enhances page loads to give us control over page transitions.
template-polyfill	A polyfill for the HTML5 <code>&lt;template&gt;</code> tag.
velocity-animate	Velocity is an animation engine with the same API as jQuery's <code>\$.animate()</code> .

## Starter Theme Dev Dependencies

Dependency	NPM Description
babel-loader	This package allows transpiling JavaScript files using Babel and webpack.
clean-webpack-plugin	A webpack plugin to remove/clean your build folder(s) before building.
css-loader	The css-loader interprets <code>@import</code> and <code>url()</code> like <code>import/require()</code> and will resolve them.
eslint	ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code.
extract-text-webpack-plugin	Extract text from a bundle, or bundles, into a separate file.
file-loader	Instructs webpack to emit the required object as file and to return its public URL
glob	Match files using the patterns the shell uses, like stars and stuff.
node-sass	Node-sass is a library that provides binding for Node.js to LibSass, the C version of the popular stylesheet

## BETA

Dependency	NPM Description	Dependency	NPM Description
	preprocessor, Sass.		CSS.
purify-css	A function that takes content (HTML/JS/PHP/etc) and CSS, and returns only the used CSS.	style-loader	Adds CSS to the DOM by injecting a <code>&lt;style&gt;</code> tag
purifycss-webpack	This plugin uses PurifyCSS to remove unused selectors from your CSS. You should use it with the <code>extract-text-webpack-plugin</code> .	uglifyjs-webpack-plugin	This plugin uses UglifyJS v3 ( <code>uglify-es</code> ) to minify your JavaScript
sass-loader	Loads a Sass/SCSS file and compiles it to	webpack	webpack is a module bundler
		url-loader	Loads files as <code>base64</code> encoded URL

# Quire Cheatsheet

## Markdown

### YAML

## Markdown

Paragraphs are made by adding two hard returns. When working in a text editor, depending on your configuration and the source of your text, lines might have hard breaks, or might all flow together. This is seen in the example below where the first paragraph has hard breaks and the second does not. In Markdwon, these will both render as single paragraphs with no hard breaks. The only way to add a hard break in markdown is with the HTML break ( `<br />` ) element.

```
 Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua
Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat.
```

```
 Lorem ipsum dolor sit amet, consectetur adip
```

Other typical Markdown formatting is listed below. Note that the spacing after the `#`, `>`,

## YAML

### Shortcodes

- and list numerals is necessary.

```
Heading 1
Heading 2
Heading 3
Heading 4
```

```
Italic Text
Bold Text
```

```
> Blockquote
```

[[Link Text](#)] (<http://www.linkadress.com>)

- dashes make
- a list with
- bullets
  - indenting
  - makes
  - sub-lists

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

## YAML

In a Quire publication, anything that could be considered data, is written in a plain-text data format called YAML.

- YAML is used for configuring how Quire works and for providing metadata about your publication in files with the extension `.yml` (both topics covered in [Metadata & Configuration](#)).
- `.yml` files are also used for storing information about figures, bibliographic references, and art objects.
- In individual pages or chapters within the publication, written in Markdown and with a `.md` extension, there is a YAML block that contains the page metadata. For example the YAML block for the page you're on right now is:

```

title: Cheatsheet
type: page

```

## YAML

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

```
- item_name:
 item_description:
```

```
item: "If the text here has a colon : or ot
```

```
item: |
 Using a pipe character, and then dropping
```

*Not all Quire YAML attributes expect Markdown style lists*

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

[YAML Lint](#) [Code Beautify](#) [validator](#)

## Shortcodes

```
{{< q-figure id="##" >}}
{{< q-figure-group id="##, ##, ##" >}}
{{< q-cite "Lastname YYYY" >}}
{{< q-bibliography >}}
{{< q-contributors range="xxxx" type="xxxx" >}}
```

BETA

## *Default Styleguide*

Markdown

[Visual Output in Quire](#)

# Default Style Guide for Quire

## Markdown

```

1 —
2 title: Style Guide
3 subtitle: Visual Output
4 weight: 201
5 type: page
6
7 # Heading 1
8 ## Heading 2
9 ### Heading 3
10 #### Heading 4
11
12 Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. Italic text in a paragraph structure. Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. Bolded text in a paragraph structure.
13
14 > This is a blockquote to designate quotes. This is a blockquote to designate quotes.
15
16 - This is a dashed list
17 - and it can keep going
18 - and be indented
19 - numerous times
20
21 1. Numbers will also
22 2. make a list
23 3. that can keep going
24
25 This is an example of a footnote.[1] This is an example of an in-text citation using Author Date.{{< q-cite "Evans 1938" >}}
26
27
 <!-- This is an HTML blank line break that can be used in Markdown. -->
28
29 {{< q-figure id="1.1" >}}
30
31 Above is a figure image with no alignment.
32
33
34 <!-- The three dashes above are a Markdown horizontal line break with an actual line instead of just spacing. -->
35
36 {{< q-figure id="1.1" class="is-pulled-left" >}}
37
38 To the left is a figure image aligned left. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around.
39
40
41 {{< q-figure id="1.1" class="is-pulled-right" >}}
42
43 To the right is a figure image aligned right. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around.
44
45
46
47 {{< q-figure-group grid="2" id="1.4, 1.5, 1.6,1.7" >}}
48
49 Above is a figure group of 4 set to grid rows of two.
50
51
52 ## Notes
53
54 [1]: This is the other part of the footnote.
55

```

markdown style guide with code

## Visual Output in Quire

### Style Guide

#### Visual Output

## Heading 1

## Heading 2

### Heading 3

### Heading 4

Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. *Italic text in a paragraph structure*. Regular unmodified text in a paragraph structure. Regular unmodified text in a paragraph structure. **Bolded text in a paragraph structure**.

This is a blockquote to designate quotes. This is a blockquote to designate quotes.

- This is a dashed list
- and it can keep going
  - and be indented
  - numerous times
- 1. Numbers will also
- 2. make a list
- 3. that can keep going

This is an example of a footnote.<sup>[1]</sup> This is an example of an in-text citation using Author Date. (Evans 1938)



Figure 1. Walker Evans. *American Legionnaire*, 1935.  
J. Paul Getty Museum

Above is a figure image with no alignment.



Figure 1. Walker Evans. *American Legionnaire*, 1935.  
J. Paul Getty Museum

To the left is a figure image aligned left. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image.



To the right is a figure image aligned right. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image. With some extra text to show how it wraps around the image.

# BETA

visual output style guide in quire

# Quire CLI Instructions

Quire CLI, or command line interface, (quire-cli) is the control for creating, previewing and outputting Quire projects. It is written in JavaScript and requires Node.js to run. Quire CLI is typically run from Terminal on a Mac, and Git Bash (or its equivalent) on a PC. The following commands are available

Command	Description
<code>quire -v</code> OR <code>quire --version</code>	Output the version number.
<code>quire -h</code> OR <code>quire --help</code>	Output usage information.
<code>quire new [project-name] [name]</code>	Create a new Quire project named <code>project-name</code> in the current directory. Name can be anything, but shouldn't contain spaces.
<code>quire preview</code>	Run a local server to preview the project in a browser. Defaults to previewing at

Command	Description
<code>quire build</code>	<code>http://localhost:1313/</code> , but will use other port numbers (such as <code>http://localhost:6532/</code> ) if <code>1313</code> is busy. The specific address will be listed in your command line terminal after running the command.
<code>quire pdf</code>	Build the files of the current project into the <code>public</code> directory. These can then be hosted on any web server.
<code>quire epub</code>	Generate a PDF version of the current project.
<code>quire install</code>	Generate an EPUB version of the current project.
<code>quire debug</code>	Install this project's theme dependencies.
	Development use only. Log info about current project.

# GitHub for Quire

[Hosting Your Project Code on GitHub](#)

[Displaying Your Project Preview on GitHub](#)

## Hosting Your Project Code on GitHub

1. Download and install GitHub Desktop: <https://desktop.github.com/>
2. Open it and go to File > Add Local Repository (Cmd-O). Select your project repository and click Add Repository
3. If there any Changes list in the left-hand pane of the window, type “Initial commit” in the Summary field below, and then click the Commit to Master button.
4. At the top right of the window, click Publish Repository (next to the cloud icon). In the pop-window, the default options should be fine, so just click Publish Repository button.

Your project code is now on GitHub. To see it, go to Repository > View on GitHub (Shift-Cmd-G) or visit <https://github.com/YOUR-USERNAME/YOUR-PROJECT-DIRECTORY-NAME>.

[Using GitHub to Manage Changes to Your Project](#)

[Using GitHub to Collaborate with Others on Your Project](#)

## Displaying Your Project Preview on GitHub

While you can host your project code on GitHub, you can also use it to host a live version of your site, the way it looks when you run `quire preview`.

1. First, follow the steps above to start a GitHub repository for your project, if you haven't already.
2. In your project folder, 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. Open Terminal and navigate to your project

## BETA

(if it's in your main user directory, just `cd YOUR-PROJECT-DIRECTORY-NAME` ), 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.)

It will take a few minutes to propagate through GitHub's system, but your site should now be published at <https://YOUR-USERNAME.github.io/YOUR-PROJECT-DIRECTORY-NAME>.

Note that if you have a GitHub account that allows for private repositories, you can have a

private repo and still run the deploy script. The code will not show up on the public GitHub site, but the preview will still be accessible at its URL. This is a good way of sharing a site preview to other collaborators during the development process, before your site is published.

## Using GitHub to Manage Changes to Your Project

TK

## Using GitHub to Collaborate with Others on Your Project

TK

## Sample publication.yml File

```

Publication.yml

#
This file houses the bibliographic info about a publication.

Title & Description

title:
subtitle:
reading_line:
short_title:

description:
 one_line:
 full:
 online:
 pdf_ebook:

promo_image:

Publication Details

url:

```

```

pub_date:
language:
pub_type: # book | journal-periodical | other

identifier:
 isbn:
 issn:
 doi:
 uuid:

series_periodical_name:
series_issue_number:

publisher:
 - name:
 location:
 url:

Contributors

contributor_as_it_appears:

contributor:
 - id:
 type: # primary | secondary | project-type
 first_name:
 last_name:

```

# BETA

```
full_name:
file_as:
title:
affiliation:
role:
role_code:
bio:
url:
pic:

Copyright & License

copyright:

license:
name:
abbreviation:
url:
icon:
scope: # full | text-only | some-exception
online_text:
pdf_ebook_text:

Formats, Resources & Links

resource_link:
- type: # other-format | related-resource

name:
url:
link_relation:
media_type:
identifier:
isbn:
issn:
doi:
uuid:
icon:

Subjects

subject:
- type: # keyword | bisac | getty
name:
identifier:

library_of_congress_cip_data:

Revision History

revision_history:
- date:
summary:

repository_url:
```

# Accessibility Principles

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 with no JavaScript and/or no CSS.

The principals 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* principals by which Quire was originally developed and that we recommend be followed by others 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.

## BETA

- 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.

# About

## History

In 2009, the Getty Foundation launched a program called the Online Scholarly Catalogue Initiative (OSCI) to support eight museums' efforts to publish their collection catalogues online. Having traditionally been published in print, collection catalogues were costly to produce, offered relatively little access (mostly through research libraries that would collect them), and were difficult, if not impossible, to update. At the end of the initiative, all eight museums had successfully published one or more collection catalogues online and had plans to do more. As discussed in the OSCI final report, however, some notable challenges remained, particularly around the *discoverability* of the catalogues after publication, and the *longevity* of the catalogues, both as individual digital objects out in the world and as publication processes internally.

Around the time OSCI was wrapping up, Getty Publications was starting to undertake our own online publishing efforts. Though we had produced a single, trial online collection catalogue during the OSCI period, our renewed effort was focused on learning the lessons from that past project as well as from OSCI. Specifically, we aimed for a way to make these types of born digital publications more discoverable and longer lasting, while doing so in a way that would be more sustainable internally. Enter static site generators, and multi-format publishing.

Static site generators like Hugo, the one under the hood in Quire, allow us to keep our content in plain text, and keep the complexity of building the site at the point of publication, rather than relying on ongoing build processes through a server that would need to be continually maintained. We could also use the static site generator (hooked together with some

other tools) to put that plain text content into formats other than online sites. Formats like PDF, e-book and even print, that would allow the publication to live where other publications live (bookstores, Amazon, Google Books, the Library of Congress, WorldCat) and thereby dramatically improve its discoverability and its archival longevity.

## Who Uses Quire?

We built Quire first for ourselves. In the last two years, we've published seven publications with early versions of Quire, and have seven more in the pipeline for the next two years. We're also starting to look at ways at expanding its usage to other projects around the Getty, potentially including the annual report, documentation and reports, newsletters, and even online exhibitions.

In its present renditions, Quire is a tool for publishing professionals and private individuals with an interest in digital publishing and everyone in between. Quire's versatility lends itself to a variety of projects and a diverse range of users.

## What is Quire's Goal?

Quire aims to streamline the digital publication process to create multi-faceted and sustainable books in multiple formats.

## What Are the Benefits of Using Quire?

As an open-source framework, Quire provides a cost-friendly alternative to digital publishing that is inclusive of a variety of features and customization that allows for the realization of projects in a user-friendly manner. Quire allows for greater preservation and digital distribution of works in a simplified format that lends itself to user accessibility.

## What Is Quire?

Quire is an open-source framework for the creation of multi-format publications. It consists of two distinct parts: 1. a set of software, and 2. a defined content model.

### 1. Quire Software

On the software side, Quire packages together a number of programs: Prince for PDF/print versions, pe-epub for EPUB, and at the heart of things, the static-site generator Hugo for the online version. The file structure, layout templates, partials and shortcodes of Quire are all Hugo conventions that have been structured to allow us create more formal digital publications (essentially, dynamic websites that make

## BETA

use of certain traditional print publication conventions like tables of contents, copyright and authorship information, linear reading orders, and the like).

### 2. Quire Content Model

The content model is documented in the API/Docs section of this guide. It defines how publication content is structured and defined in Quire (including data on the publication itself, pages of the publication, contributors, figures, bibliography, etc.) and how the Quire software templates use this structured content. The content model is designed to be as independent of the Quire software stack as possible, so that long-term, it might be used in other ways. If, for example, any part of the Quire software were replaced with something new, or if the content of a publication were to be used for an entirely different use.

[Full size version of the content model](#)

## What Do the Quire Repositories Do?

While conceptually, Quire is made of the two distinct parts defined above, these play out in multiple working parts in actual practice.

`quire-cli`

Quire CLI, or command line interface, (`quire-cli`) is the control for creating, previewing and outputting Quire projects. It is written in JavaScript and requires Node.js 8.9.4 LTS to run. Quire CLI is typically run from Terminal on a Mac, and Git Bash (or its equivalent) on a PC. The following commands are available:

Command	Description
<code>quire -V</code> or <code>quire --version</code>	Output the version number.
<code>quire -h</code> or <code>quire --help</code>	Output usage information.
<code>quire new</code> <code>project-</code> <code>name</code>	Create a new Quire project named <code>project-name</code> in the current directory. Name can be anything, but shouldn't contain spaces.
<code>quire preview</code>	Run a local server to preview the project in a browser. Defaults to previewing at <code>http://localhost:1313/</code> , but will use other port numbers (such as <code>http://localhost:6532/</code> ) if <code>1313</code> is busy. The specific address will be listed in your command line terminal after running the command.

# BETA

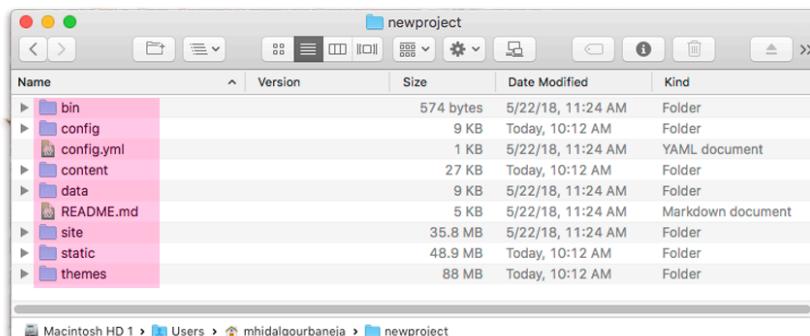
## Command Description

quire	Build the files of the current project into the <code>public</code> directory. These can
build	then be hosted on any web server.
quire	Generate a PDF version of the current project.
pdf	
quire	Generate an EPUB version of the current project.
epub	

Read more in [Install and Uninstall](#), and [Multi-Format Output](#).

## quire-starter

Quire Starter (`quire-starter`) is a starter content repository used as placeholder content when starting a new Quire project with the `quire new` command. It comes with some pre-defined example content and pages with which to get started.



## Project Folder

File structure

## quire-starter-theme

Quire Starter Theme (`quire-starter-theme`) is the basic theme that is included when starting a new Quire project with the `quire new` command. It is designed to broadly cover a full range of use-cases and to demonstrate the range of Quire content model, without being too overly specific or complex, making it easy to customize and build from. Read more about the use of themes in [Customizing Styles](#).

# BETA

Name	Date Modified	Size	Kind
quire-starter-theme	Today at 3:56 PM	--	Folder
└── archetypes	Today at 11:07 AM	--	Folder
└── default.md	Today at 11:07 AM	8 bytes	Markdo...ument
└── layouts	Today at 11:07 AM	--	Folder
└── _default	Today at 11:07 AM	--	Folder
└── 404.html	Today at 11:07 AM	Zero bytes	HTML
└── contents	Today at 11:07 AM	--	Folder
└── cover	Today at 11:07 AM	--	Folder
└── entry	Today at 11:07 AM	--	Folder
└── essay	Today at 11:07 AM	--	Folder
└── partials	Today at 11:07 AM	--	Folder
└── section-head	Today at 11:07 AM	--	Folder
└── shortcodes	Today at 11:07 AM	--	Folder
└── LICENSE.md	Today at 11:07 AM	1 KB	Markdo...ument
└── node_modules	Today at 11:09 AM	--	Folder
└── package-lock.json	Today at 11:07 AM	187 KB	JSON file
└── package.json	Today at 11:07 AM	1 KB	JSON file
└── README.md	Today at 11:07 AM	1 KB	Markdo...ument
└── source	Today at 11:07 AM	--	Folder
└── static	Today at 11:07 AM	--	Folder
└── theme.toml	Today at 11:07 AM	454 bytes	TOML file
└── webpack.config.js	Today at 11:07 AM	2 KB	JavaSc...t script

Starter Theme

More themes will be added in the future.

quire

Quire (quire) is the repository for the guide and documentation you are currently reading.