Introduction

Overview

This site provides documentation, training, and other notes for the Jekyll Documentation theme. There's a lot of information about how to do a variety of things here, and it's not all unique to this theme. But by and large, understanding how to do things in Jekyll depends on how your theme is coded. As a result, these additional details are provided.

The instructions here are geared towards technical writers working on documentation. You may have a team of one or more technical writers working on documentation for multiple projects. You can use this same theme to author all of your documentation for each of your products. The theme is set up to push out documentation for multiple projects all from the same source. You can also share content across projects.

Survey of features

Some of the more prominent features of this theme include the following:

- Bootstrap framework
- Sidebar for table of contents
- Top navigation bar with drop-down menus
- PDF generation (through Prince XML utility)
- Build scripts to automate the workflow
- Notes, tips, and warning information notes
- A nifty system for creating links to different pages
- Tags for alternative nativation
- Content sharing across projects
- Emphasis on pages, not posts
- Relative (rather than absolute) link structure, so you can push the outputs anywhere and easily view them

I'm using this theme for my documentation projects, building about 20 different outputs for various products, versions, languages, and audiences from the same set of files. This single sourcing requirement has influenced how I constructed this theme.

For more discussion about the available features, see .

Getting started

To get started, see these three topics:

1.
2.
3.

PDF Download Option for Help Material

If you would like to download this help file as a PDF, you can do so here. The PDF is comprehensive of all the content in the online help.

PDF DOWNLOAD

The PDF contains a timestamp in the header indicating when it was last generated.

Supported features

Summary: If you're not sure whether Jekyll and this theme will support your requirements, this list provides a semi-comprehensive overview of available features.

Before you get into exploring Jekyll as a potential platform for help content, you may be wondering if it supports some basic features. The following table shows what is supported in Jekyll and this theme.

FEATURES	SUPPORTED	NOTES
Content re-use	Yes	Supports re-use through Liquid. You can re-use variables, snippets of code, entire pages, and more. In DITA speak, this includes conref and keyref.
Markdown	Yes	You can author content using Markdown syntax. This is a wiki-like syntax for HTML that you can probably pick up in 10 minutes. Where Mark-down falls short, you can use HTML. Where HTML falls short, you use Liquid, which is a scripting that allows you to incorporate more ad-vanced logic.
Responsive design	Yes	Uses Bootstrap framework.
Translation	Yes	I haven't done a translation project yet (just a pi-lot test). Here's the basic approach: Export the pages and send them to a translation agency. Then create a new project for that language and insert the translated pages. Everything will be translated.

FEATURES	SUPPORTED	NOTES
PDF	Yes	You can generate PDFs from your Jekyll site. This theme uses Prince XML (costs $495) to do the PDF conversion task. You basically set up a page that uses Liquid logic to get all the pages you want, and then you use PrinceXML (not part of Jekyll) to convert that page into a PDF.
Collaboration	Yes	You collaborate with Jekyll projects the same way that developers collaborate with software projects. (You don't need a CMS.) Because you're working with text file formats, you can use any version control software (Git, Mercurial, Perforce, Bitbucket, etc.) as a CMS for your files.
Scalability	Yes	Your site can scale to any size. It's up to you to determine how you will design the information architecture for your thousands of pages. You can choose what you display at first, second, third, fourth, and more levels, etc. Note that when your project has thousands of pages, the build time will be longer (maybe 1 minute per thousand pages?). It really depends on how many for loops you have iterating through the pages.
Lightweight architecture	Yes	You don't need a LAMP stack (Linux, Apache, MySQL, PHP) architecture to get your site running. All of the building is done on your own machine, and you then push the static HTML files onto a server.
Multichannel output	Yes	This term can mean a number of things, but let's say you have 10 different sites you want to generate from the same source. Maybe you have 7 different versions of your product, and 3 different locations. You can assemble your Jekyll site with various configurations, variants, and more. Jekyll actually does all of this quite well. Just specify a different config file for each unique build.

FEATURES	SUPPORTED	NOTES
Skinnability	Yes	You can skin your Jekyll site to look identical to pretty much any other site online. If you have a UX team, they can really skin and design the site using all the tools familiar to the modern designer -- JavaScript, HTML5, CSS, jQuery, and more. Jekyll is built on the modern web development stack rather than the XML stack (XSLT, XPath, XQuery).
Support	Yes	The community for your Jekyll site isn't so much other tech writers (as is the case with DITA) but rather the wider web development community. Jekyll Talk (http://talk.jekyllrb.com) is a great resource. So is Stack Overflow.
Blogging features	No	This theme just uses pages, not posts. I may integrate in post features in the future, but the theme really wasn't designed with posts in mind. If you want a post version of the site, you can clone my blog theme (https://github.com/tomjohnson1492/tomjohnson1492.github.io), which is highly similar in that it's based on Bootstrap, but it uses posts to drive most of the features. I wanted to keep the project files simple.
CMS interface	No	Unlike with WordPress, you don't log into an interface and navigate to your files. You work with text files and preview the site dynamically in your browser. Don't worry -- this is part of the simplicy that makes Jekyll awesome. I recommend using WebStorm as your text editor.
WYSIWYG interface	No, but ...	As noted in the previous point, I use WebStorm to author content, because I like working in text file formats. But you can use any Markdown editor you want (e.g., Lightpaper for Mac, Marked) to author your content.

FEATURES	SUPPORTED	NOTES
Versioning	Yes, but...	Jekyll doesn't version your files. You upload your files to a version control system such as Git. Your files are versioned there.
PC platform	Yes, but ...	Jekyll isn't officially supported on Windows, and since I'm on a Mac, I haven't tried using Jekyll on Windows. See this page in Jekyllrb help (http://jekyllrb.com/docs/windows/) for details about installing and running Jekyll on a Windows machine. A couple of Windows users who have contacted me have been unsuccessful in installing Jekyll on Windows, so beware. In the configuration files, use `rouge` instead of `pygments` (which is Python-based) to avoid conflicts.
jQuery plugins	Yes	You can use any jQuery plugins you and other JavaScript, CMS, or templating tools. However, note that if you use Ruby plugins, you can't directly host the source files on Github Pages because Github Pages doesn't allow Ruby plugins. Instead, you can just push your output to any web server. If you're not planning to use Github Pages, there are no restrictions on any plugins of any sort. Jekyll makes it super easy to integrate every kind of plugin imaginable. This theme doesn't actually use any plugins, so you can publish on Github if you want.
Bootstrap integration	Yes	This theme is built on Bootstrap (http://getbootstrap.com/). If you don't know what Bootstrap is, basically this means there are hundreds of pre-built components, styles, and other elements that you can simply drop into your site. For example, the responsive quality of the site comes about from the Bootstrap code base.

FEATURES	SUPPORTED	NOTES
Fast-loading pages	Yes	This is one of the Jekyll's strengths. Because the files are static, they loading extremely fast, approximately 0.5 seconds per page. You can't beat this for performance. (A typically database-driven site like WordPress averages about 2.5 + seconds loading time per page.) Because the pages are all static, it means they are also extremely secure. You won't get hacked like you might with a WordPress site.
Relative links	Yes	This theme is built entirely with relative links, which means you can easily move the files from one folder to the next and it will still display. You don't need to view the site on a web server either -- you can view it locally just clicking the files. This relative link structure facilitates scenarios where you need to archive versions of content or move the files from one directory (a test directory) to another (such as a production directory).
Themes	Yes	You can have different themes for different outputs. If you know CSS, theming both the web and print outputs is pretty easy.
Open source	Yes	This theme is entirely open source. Every piece of code is open, viewable, and editable. Note that this openness comes at a price — it's easy to make changes that break the theme or otherwise cause errors.

Pages

Summary: This theme uses pages only, not posts. You need to make sure your pages have the appropriate frontmatter. One frontmatter tag your users might find helpful is the summary tag. This functions similar in purpose to the shortdesc element in DITA.

Where to author content

Use a text editor such as Sublime Text, WebStorm, IntelliJ, or Atom to create pages.

My preference is IntelliJ/WebStorm, since it will treat all files in your project as belonging to a project. This allows you to easily search for instances of keywords, do find-and-replace operations, or do other actions that apply across the whole project.

Page names and excluding files from outputs

By default, everything in your project is included in the output. This is problematic when you're single sourcing and need to exclude some files from an output.

Here's the approach I've taken. Put all files in your root directory, but put the project name first and then any special conditions. For example, thirtybees_writers_intro.md.

In your configuration file, you can exclude all files that don't belong to that project by using wildcards such as the following:

exclude:

- thirtybees_*
- thirtybees_writers_*

These wildcards will exclude every match after the `*`.

Frontmatter

Make sure each page has frontmatter at the top like this:

```
---
title: Your page title
tags: [formatting, getting_started]
keywords: overview, going live, high-level
last_updated: November 30, 2015
summary: "Deploying DeviceInsight requires the following steps."
---
```

Frontmatter is always formatted with three hyphens at the top and bottom. Your frontmatter must have a `title` value. All the other values are optional.

The following table describes each of the frontmatter that you can use with this theme:

FRONTMATTER	REQUIRED?	DESCRIPTION
title	Required	The title for the page
tags	Optional	Tags for the page. Make all tags single words, with hyphens if needed. Separate them with commas. Enclose the whole list within brackets. Also, note that tags must be added to _data/tags_doc.yml to be allowed entrance into the page.
keywords	Optional	Synonyms and other keywords for the page. This information gets stuffed into the page's metadata to increase SEO. The user won't see the keywords, but if you search for one of the keywords, it will be picked up by the search engine.
last_updated	Optional	The date the page was last updated. This information could helpful for readers trying to evaluate how current and authoritative information is. If included, the last_updated date appears in the footer of the page.

FRONTMATTER	REQUIRED?	DESCRIPTION
summary	Optional	A 1-2 word sentence summarizing the content on the page. This gets formatted into the summary section in the page layout. Adding summaries is a key way to make your content more scannable by users (check out Jakob Nielsen's site (http://www.nngroup.com/articles/corporate-blogs-front-page-structure/) for a great example of page summaries.)
datatable	Optional	Boolean. If you add `true`, then scripts for the jQuery datatables plugin (https://www.datatables.net/) get included on the page.
video	Optional	If you add `true`, then scripts for Video JS: The HTML5 video player (http://www.videojs.com/) get included on the page.

☑ **Tip:** You can see the scripts that conditionally appear by looking in the _layouts/default.html page. Note that these scripts are served via a CDN, so the user must be online for the scripts to work. However, if the user isn't online, the tables and video still appear. In other words, they degrade gracefully.

What about permalinks?

What about permalinks? This theme isn't build using permalinks because it makes linking and directory structures problematic. Permalinks generate an index file inside a folder for each file in the output. This makes it so links (to other pages as well as to resources such as styles and scripts) need to include `../` depending upon where the other assets are located. But for any pages outside folders, such as the index.html page, you wouldn't use the `../` structure.

Basically, permalinks complicate the linking structure significantly, so they aren't used here. As a result, page URLs have an .html extension. If you include `permalink: something` in your frontmatter, your link to the page will break (actually, you could still go to sample instead of sample.html, but none of the styles or scripts will be correctly referenced).

Colons in page titles

If you want to use a colon in your page title, you must enclose the title's value in quotation marks.

Saving pages as drafts

If you add `published: false` in the frontmatter, your page won't be published. You can also move draft pages into the _drafts folder to exclude them from the build.

> ✓ **Tip:** You can create file templates in WebStorm that have all your common frontmatter, such as all possible tags, prepopulated. See for details.

Markdown or HTML format

Pages can be either Markdown or HTML format (specified through either an .md or .html file extension).

If you use Markdown, you can also include HTML formatting where needed. But not vice versa — if you use HTML (as your file extension), you can't insert Markdown content.

Also, if you use HTML inside a Markdown file, you cannot use Markdown inside of HTML. But you can use HTML inside of Markdown.

For your Markdown files, note that a space or two indent will set text off as code or blocks, so avoid spacing indents unless intentional.

Where to save pages

Store all your pages inside the root directory. This is because the site is built with relative links. There aren't any permalinks or baseurls used in the link architecture. This relative link nature of the site allows you to easily move it from one folder to another without invalidating the links.

If this approach creates too many files in one long list, consider grouping files into Favorites sections using WebStorms Add to Favorites feature.

Github-flavored Markdown

You can use standard Multimarkdown syntax for tables. You can also use fenced code blocks. The configuration file shows the Markdown processor and extensiosn:

```
markdown: redcarpet

redcarpet:
  extensions: ["no_intra_emphasis", "fenced_code_blocks", "tables", "with_to
c_data"]
```

These extensions mean the following:

REDCARPET EXTENSION	DESCRIPTION
no_intra_emphasis	don't italicize words with underscores
fenced_code_blocks	allow three backticks before and after code blocks instead of `<pre>` tags
tables	allow table syntax
with_toc_data	add ID tags to headings automatically

You can also add "autolink" as an option if you want links such as http://google.com to automatically be converted into links.

Note: Make sure you leave the `with_toc_data` option included. This auto-creates an ID for each Markdown-formatted heading, which then gets injected into the mini-TOC. Without this auto-creation of IDs, the mini-TOC won't include the heading. If you ever use HTML formatting for headings, you need to manually add an ID attribute to the heading in order for the heading to appear in the mini-TOC.

Automatic mini-TOCs

By default, a mini-TOC appears at the top of your pages and posts. If you don't want this, you can remove the `{% include toc.html %}` from the layouts/page.html file.

If you don't want the TOC to appear for a specific page, add `toc: false` in the frontmatter of the page.

The mini-TOC requires you to use the `##` syntax for headings. If you use `<h2>` elements, then you must add an ID attribute for the h2 element in order for it to appear in the mini-TOC.

Specify a particular page layout

The configuration file sets the default layout for pages as the "page" layout.

You can create other layouts inside the layouts folder. If you create a new layout, you can specify that your page use your new layout by adding `layout: mylayout.html` in the page's frontmatter. Whatever layout you specify in the frontmatter of a page will override the layout default set in the configuration file.

Comments

Disqus, a commenting system, is integrated into the theme. In the configuration file, specify the Disqus code for the universal code, and Disqus will appear. If you don't add a Disqus value, the Disqus code isn't included.

Posts

This theme isn't coded with any kind of posts logic. For example, if you wanted to add a blog to your project that leverages posts, you couldn't do this with the theme. However, you could easily take the post logic from another site and integrate it into this theme. I've just never had a strong need to integrate blog posts into documentation.

Custom keyboard shortcuts

Some of the Jekyll syntax can be slow to create. Using a utility such as aText (https://www.trankynam.com/atext/) can make creating content a lot of faster.

For example, when I type `jif`, aText replaces it with `{% if site.platform == "x" %}`. When I type `jendif`, aText replaces it with `{% endif %}`.

You get aText from the App Store on a Mac for about $5.

There are alternatives to aText, such as Typeitforme. But aText seems to work the best. You can read more about it on Lifehacker (http://lifehacker.com/5843903/the-best-text-expansion-app-for-mac).

WebStorm Text Editor

Summary: You can use a variety of text editors when working with a Jekyll project. WebStorm from IntelliJ offers a lot of project-specific features, such as find and replace, that make it ideal for working with tech comm projects.

About text editors and WebStorm

There are a variety of text editors available, but I like WebStorm the best because it groups files into projects, which makes it easy to find all instances of a text string, to do find and replace operations across the project, and more.

If you decide to use WebStorm, here are a few tips on configuring the editor.

Remove unnecessary plugins

By default, WebStorm comes packaged with a lot more functionality than you probably need. You can lighten the editor by removing some of the plugins. Go to **WebStorm > Preferences > Plugins** and clear the check boxes of plugins you don't need.

Add the Markdown Support plugin

Since you'll be writing in Markdown, having color coding and other support for Markdown is key. Install the Markdown Support plugin by going to **WebStorm > Preferences > Plugins** and clicking **Install JetBrains Plugin**. Search for **Markdown Support**.

Learn a few key commands

COMMAND	SHORTCUTS
Shift + Shift	Allows you to find a file by searching for its name.

COMMAND	SHORTCUTS
Shift + Command + F	Find in whole project. (WebStorm uses the term "Find in path".)
Shift + Command + R	Replace in whole project. (Again, WebStorm calls it "Replace in path.")
Command + F	Find on page
Shift + R	Replace on page
Right-click > Add to Favorites	Allows you to add files to a Favorites section, which expands below the list of files in the project pane.
Shift + tab	Applies outdenting (opposite of tabbing)
Shift + Function + F6	Rename a file
Command + Delete	Delete a file
Command + 2	Show Favorites pane
Shift + Option + F	Add to Favorites

☑ **Tip:** If these shortcut keys aren't working for you, make sure you have the "Max OS X 10.5+" keymap selected. Go to **WebStorm > Preferences > Keymap** and select it there.

Identifying changed files

When you have the Git and Github integration, changed files appear in blue. This lets you know what needs to be committed to your repository.

Creating file templates

Rather than insert the frontmatter by hand each time, it's much faster to simply create a Jekyll template. To create a Jekyll template in WebStorm:

1. Right-click a file in the list of project files, and select **New > Edit File Templates**.

   If you don't see the Edit File Templates option, you may need to create a file template first. Go to **File > Default Settings > Editor > File and Code Templates**. Create a new file template with an md extension, and then close and restart WebStorm. Then repeat this step and you will see the File Templates option appear in the right context menu.

2. In the upper-left corner of the dialog box that appears, click the **+** button to create a new template.

3. Name it something like Jekyll page. Insert the frontmatter you want, and save it.

   To use the Jekyll template, when you create a new file in your WebStorm project, you can select your Jekyll file template.

Disable pair quotes

By default, each time you type `'`, WebStorm will pair the quote (creating two quotes). You can disable this by going to **WebStorm > Preferences > Editor > Smartkeys**. Clear the **Insert pair quotes** check box.

Conditional logic

Summary: You can implement advanced conditional logic that includes if statements, or statements, unless, and more. This conditional logic facilitates single sourcing scenarios in which you're outputting the same content for different audiences.

About Liquid and conditional statements

If you want to create different outputs for different audiences, you can do all of this using a combination of Jekyll's Liquid markup and values in your configuration file.

You can then incorporate conditional statements that check the values in the configuration files.

☑ **Tip:** Definitely check out Liquid's documentation (http://docs.shopify.com/themes/liquid-documentation/basics) for more details about how to use operators and other liquid markup. The notes here are a small, somewhat superficial sample from the site.

Where to store filtering values

You can filter content based on values that you have set either in your config file or in a file in your _data folder. If you set the attribute in your config file, you need to restart the Jekyll server to see the changes. If you set the value in a file in your _data folder, you don't need to restart the server when you make changes.

Required conditional attributes

This theme requires you to add the following attributes in your configuration file:

- project
- audience
- product
- platform

- version

If you've ever used DITA, you probably recognize these attributes, since DITA has mostly the same ones. I've found that most single_sourcing projects I work on can be sliced and diced in the ways I need using these conditional attributes.

If you're not single sourcing and you find it annoying having to specify these attributes in your sidebar, you can rip out the logic from the sidebar.html, topnav.html file and any other places where conditions.html appears; then you wouldn't need these attributes in your configuration file.

Conditional logic based on config file value

Here's an example of conditional logic based on a value in the configs/ config_writer.yml file. In my config_writer.yml file, I have the following:

```
audience: writers
```

On a page in my site (it can be HTML or markdown), I can conditionalize content using the following:

```
{% if site.audience == "writers" %}
The writer audience should see this...
{% elsif site.audience == "designers" %}
The designer audience should see this ...
{% endif %}
```

This uses simple `if-elsif` logic to determine what is shown (note the spelling of `elsif`). The `else` statement handles all other conditions not handled by the `if` statements.

Here's an example of `if-else` logic inside a list:

```
To bake a casserole:

1. Gather the ingredients.
{% if site.audience == "writer" %}
2. Add in a pound of meat.
{% elsif site.audience == "designer" %}
3. Add in an extra can of beans.
{% endif %}
3. Bake in oven for 45 min.
```

You don't need the `elsif` or `else`. You could just use an `if` (but be sure to close it with `endif`).

Or operator

You can use more advanced Liquid markup for conditional logic, such as an `or` command. See Shopify's Liquid documentation (http://docs.shopify.com/themes/liquid-documentation/basics/operators) for more details.

For example, here's an example using `or`:

```
{% if site.audience contains "vegan" or site.audience == "vegetarian" %}
    // run this.
{% endif %}
```

Note that you have to specify the full condition each time. You can't shorten the above logic to the following:

```
{% if site.audience contains "vegan" or "vegetarian" %}
    // run this.
{% endif %}
```

This won't work.

Unless operator

You can also use `unless` in your logic, like this:

```
{% unless site.output == "pdf" %}
...
{% endunless %}
```

When figuring out this logic, read it like this: "Run the code here *unless* this condition is satisfied." Or "If this condition is satisfied, don't run this code."

Don't read it the other way around or you'll get confused. (It's not executing the code only if the condition is satisfied.)

In this situation, if `site.print == true`, then the code will *not* be run here.

Storing conditions in the _data folder

Here's an example of using conditional logic based on a value in a data file:

```
{% if site.data.options.output == "alpha" %}
show this content...
{% elsif site.data.options.output == "beta" %}
show this content...
{% else %}
this shows if neither of the above two if conditions are met.
{% endif %}
```

To use this, I would need to have a _data folder called options where the `output` property is stored.

I don't really use the _data folder as much for project options. I store them in the configuration file because I usually want different projects to use different values for the same property.

For example, maybe a file or function name is called something different for different audiences. I currently single source the same content to at least two audiences in different markets.

For the first audience, the function name might be called `generate`, but for the second audience, the same function might be called called `expand`. In my content, I'd just use `{{site.function}}`. Then in the configuration file I change its value appropriately for the audience.

Specifying the location for _data

You can also specify a `data_source` for your data location in your configuration file. Then you aren't limited to simply using `_data` to store your data files.

For example, suppose you have 2 projects: alpha and beta. You might store all the data files for alpha inside data_alpha, and all the data files for beta inside data_beta.

In your alpha configuration file, specify the data source like this:

```
data_source: data_alpha
```

Then create a folder called _data_alpha.

For your beta configuratoin file, specify the data source like this:

```
data_source: data_beta
```

Then create a folder called _data_beta.

Conditional logic based on page namespace

You can also create conditional logic based on the page namespace. For example, create a page with front matter as follows:

```
---
layout: page
user_plan: full
---
```

Now you can run logic based on the conditional property in that page's front matter:

```
{% if page.user_plan == "full" %}
// run this code
{% endif %}
```

Conditions versus includes

If you have a lot of conditions in your text, it can get confusing. As a best practice, whenever you insert an `if` condition, add the `endif` at the same time. This will reduce the chances of forgetting to close the if statement. Jekyll won't build if there are problems with the liquid logic.

If your text is getting busy with a lot of conditional statements, consider putting a lot of content into includes so that you can more easily see where the conditions begin and end.

Content reuse

Summary: You can reuse chunks of content by storing these files in the includes folder. You then choose to include the file where you need it. This works similar to conref in DITA, except that you can include the file in any content type.

About content reuse

You can embed content from one file inside another using includes. Put the file containing content you want to reuse (e.g., mypage.html) inside the _includes/thirtybees folder (replacing "thirtybees" with your project's name), and then use a tag like this:

```
{% include thirtybees/mypage.html %}
```

With content in your _includes folder, you don't add any frontmatter to these pages because they will be included on other pages already containing frontmatter.

Also, when you include a file, all of the file's contents get included. You can't specify that you only want a specific part of the file included. However, you can use parameters with includes. See Jekyll's documentation (http://stackoverflow.com/questions/21976330/passing-parameters-to-inclusion-in-liquid-templates) for more information on that.

Re-using content across projects

When you want to re-use a topic across projects, store the content in the \includes folder (it can be in any project's subfolder). Any folder that begins with an underscore (_) isn't included in the site output.

Also be sure to put any images in the common_images folder. None of the assets in the common_images folder should be excluded in the configuration files. This means every project's output will include the resources from the common_images folder.

However, each project will likely exclude content from the specific folders where the pages are stored. This is why reuse across projects requires you to use the _includes folder and the common_images folder. (Unfortunately you can't include an image from the _includes folder.)

Page-level variables

You can also create custom variables in your frontmatter like this:

```
---
title: Page-level variables
permalink: /page_level_variables/
thing1: Joe
thing2: Dave
---
```

You can then access the values in those custom variables using the `page` namespace, like this:

```
thing1: {{page.thing1}}
thing2: {{page.thing2}}
```

I haven't found a use case for page-level variables, but it's nice to know they're available.

I use includes all the time. Most of the includes in the _includes directory are pulled into the theme layouts. For those includes that change, I put them inside custom and then inside a specific project folder.

Collections

Summary: Collections are useful if you want to loop through a special folder of pages that you make available in a content API. You could also use collections if you have a set of articles that you want to treat differently from the other content, with a different layout or format.

What are collections

Collections are custom content types different from pages and posts. You might create a collection if you want to treat a specific set of articles in a unique way, such as with a custom layout or listing. For more detail on collections, see Ben Balter's explanation of collections here (http://ben.balter.com/2015/02/20/jekyll-collections/).

Create a collection

To create a collection, add the following in your configuration file:

```
collections:
  tooltips:
    output: true
```

In this example, "tooltips"" is the name of the collection.

Interacting with collections

You can interact with collections by using the `site.collectionname` namespace, where `collectionname` is what you've configured. In this case, if I wanted to loop through all tooltips, I would use `site.tooltips` instead of `site.pages` or `site.posts` .

See Collections in the Jekyll documentation (http://jekyllrb.com/docs/collections/) for more information.

How to use collections

I haven't found a huge use for collections in normal documentation. However, I did find a use for collections in generating a tooltip file that would be used for delivering tooltips to a user interface from text files in the documentation. See for details.