Multiple product versions - Balsamiq, Leon Barnard

by Leon Barnard
April 4, 2017

CATEGORIES articles TAGS case study, balsamiq, static sites, use case, wireframes, GitHub, docs, repos, hugo, tools, gif, animated gifs

Use Case Part I: Balsamiq

Leon Barnard, Designer and Writer (and Developer-in-training) at Balsamiq, describes how the second version of the Balsamiq Docs static site went beyond purely functional to do things the team didn't think Markdown and static sites could do.

In this three-part series, Leon describes some great solutions to problems that many technical sites have faced.

The three parts cover documenting multiple product versions, adding play and pause to animated GIFs, and providing flexibility and expansion options in lists of linked pages.

Background

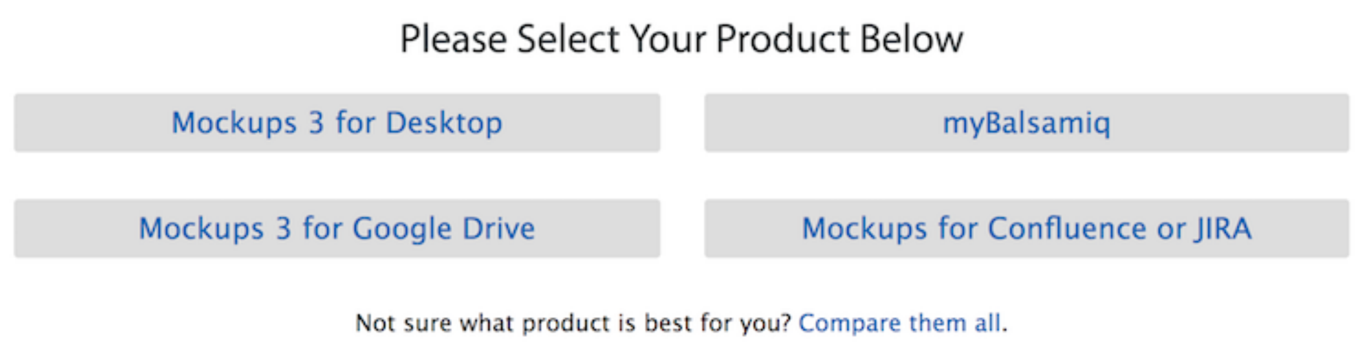
A year and a half ago we ditched our flaky content management system and converted our documentation site over to a "docs like code" system using Hugo, Gulp, and GitHub, with content written in Markdown. It was a long process and we were happy just to see it up and running.

After the dust settled, we started imagining what we wanted for the next version, and realized that the system we had built had limitations that we would have to overcome. This is a story of three challenges and how we solved them.

Note: *The code in this article is specific to Hugo, the static site generator we use, but should be adaptable to other static site generators.*

Challenge #1: Documenting multiple product versions

Our product, Balsamiq Mockups, comes in multiple "flavors," as we often call them. There are two versions of the core product (the wireframe editor) that are sold as: a stand-alone desktop version, a hosted web version ("myBalsamiq"), and plugins for Google Drive and Atlassian Confluence and JIRA.

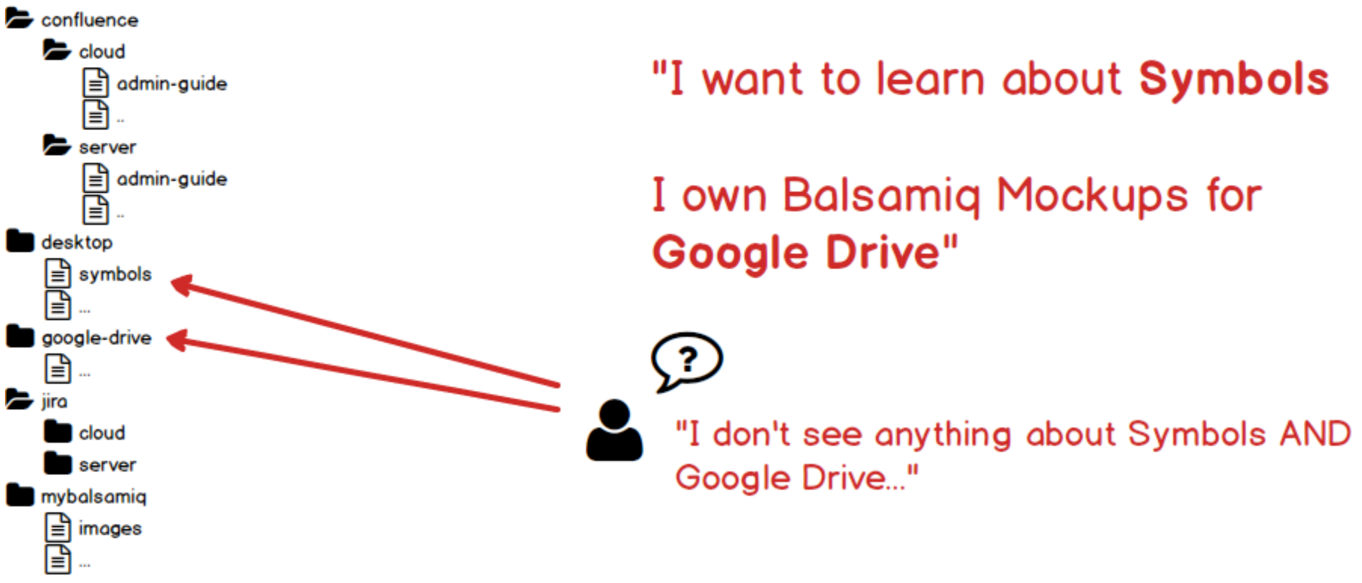

Please Select Your Product Below
Mockups 3 for Desktop
myBalsamiq
Mockups 3 for Google Drive
Mockups for Confluence or JIRA
Not sure what product is best for you? Compare them all.

That's seven different products that we sell, catered to different types of users. This is great for our customers, but a challenge for our docs.

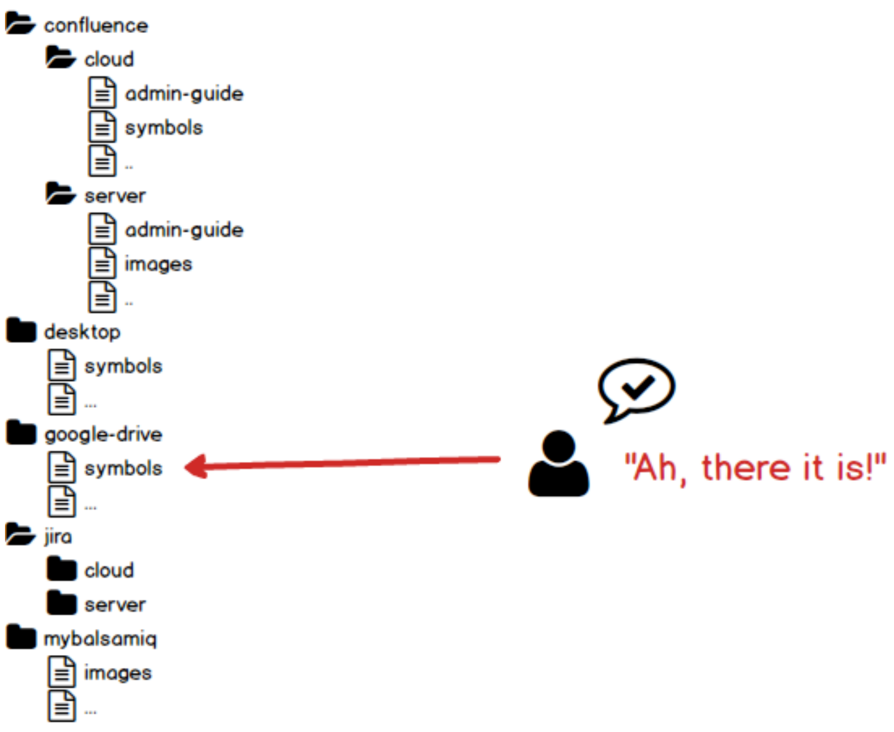
The majority of our documentation is on how to use one of the two core wireframe editors. Very little has to do with platform-specific features. So, seven products that we sell, but only two main sets of documentation.

For historical reasons, most of the common documentation is in the Desktop version docs. If you used our product on a different platform, you had to know to go to the Desktop docs.

This is what the content folder in our GitHub repo looked like, which meant that a lot of customers had this experience:


confluence
cloud
admin-guide
..
server
admin-guide
..
desktop
symbols
...
google-drive
...
jira
cloud
server
mybalsamiq
images
...
"I want to learn about Symbols
I own Balsamiq Mockups for Google Drive"
?
"I don't see anything about Symbols AND Google Drive..."

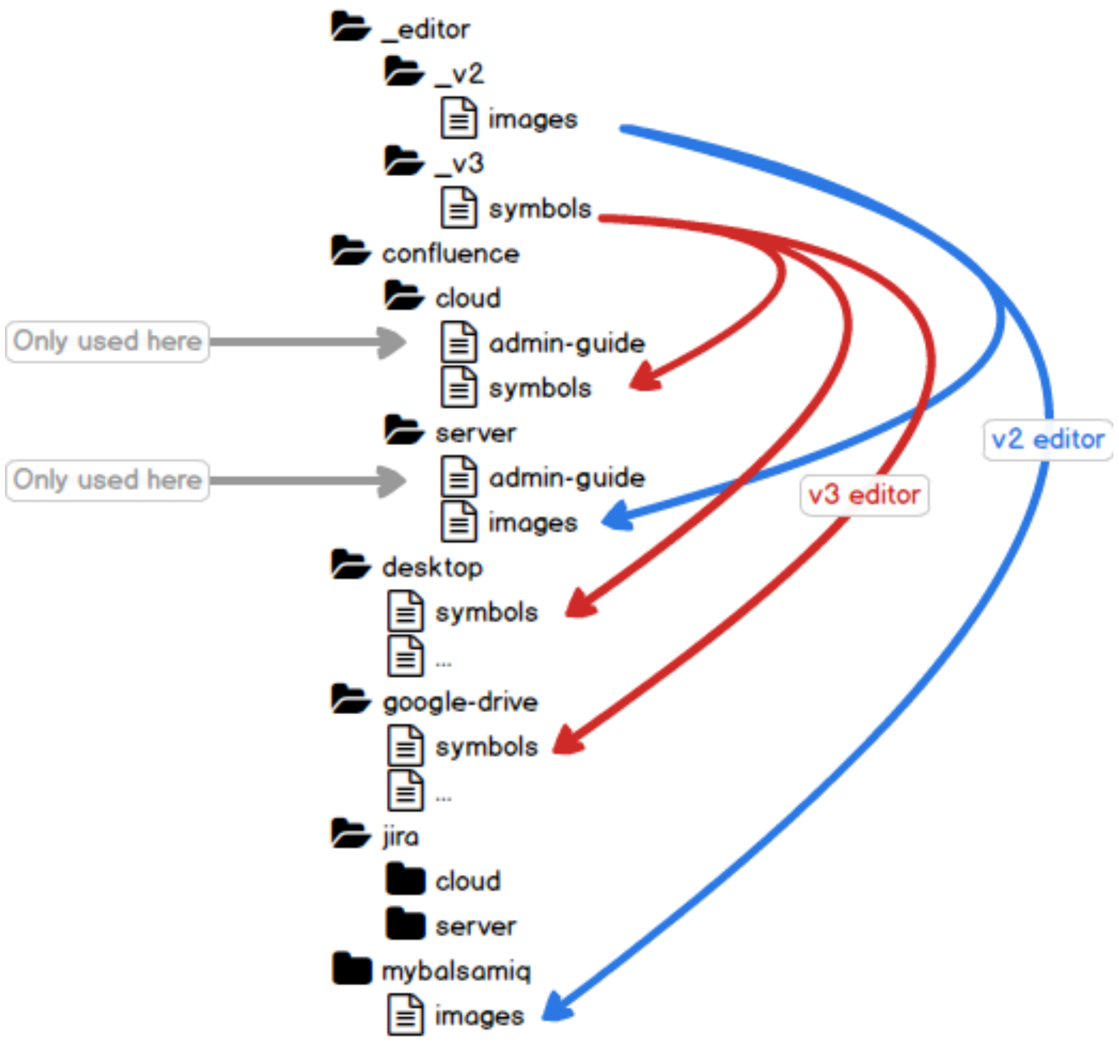
For the next update to our docs, we decided that we needed to finally address this issue. We wanted a complete set of docs for each product version and platform.



But, of course, we didn't want to maintain seven copies of each article. We wanted to be able to write the docs once and have them show up across multiple versions.

A simple solution, I suppose, would have been to switch from Markdown to a file format that supports including content from one file in another (like reStructuredText or AsciiDoc). **But, I'm stubborn** (and Hugo doesn't support them well). I like how simple Markdown is and I wanted to see if we could achieve what we wanted without abandoning it.

I enlisted the help of our resident Go expert, Luis, and he wrote some code to do just what we wanted. This code inside one of the Hugo templates allowed us to put help files in two hidden folders (called "_v2" and "_v3") that would get injected into files across multiple documentation folders each time the site was generated. The result was that users could now find the information where they expected to find it.



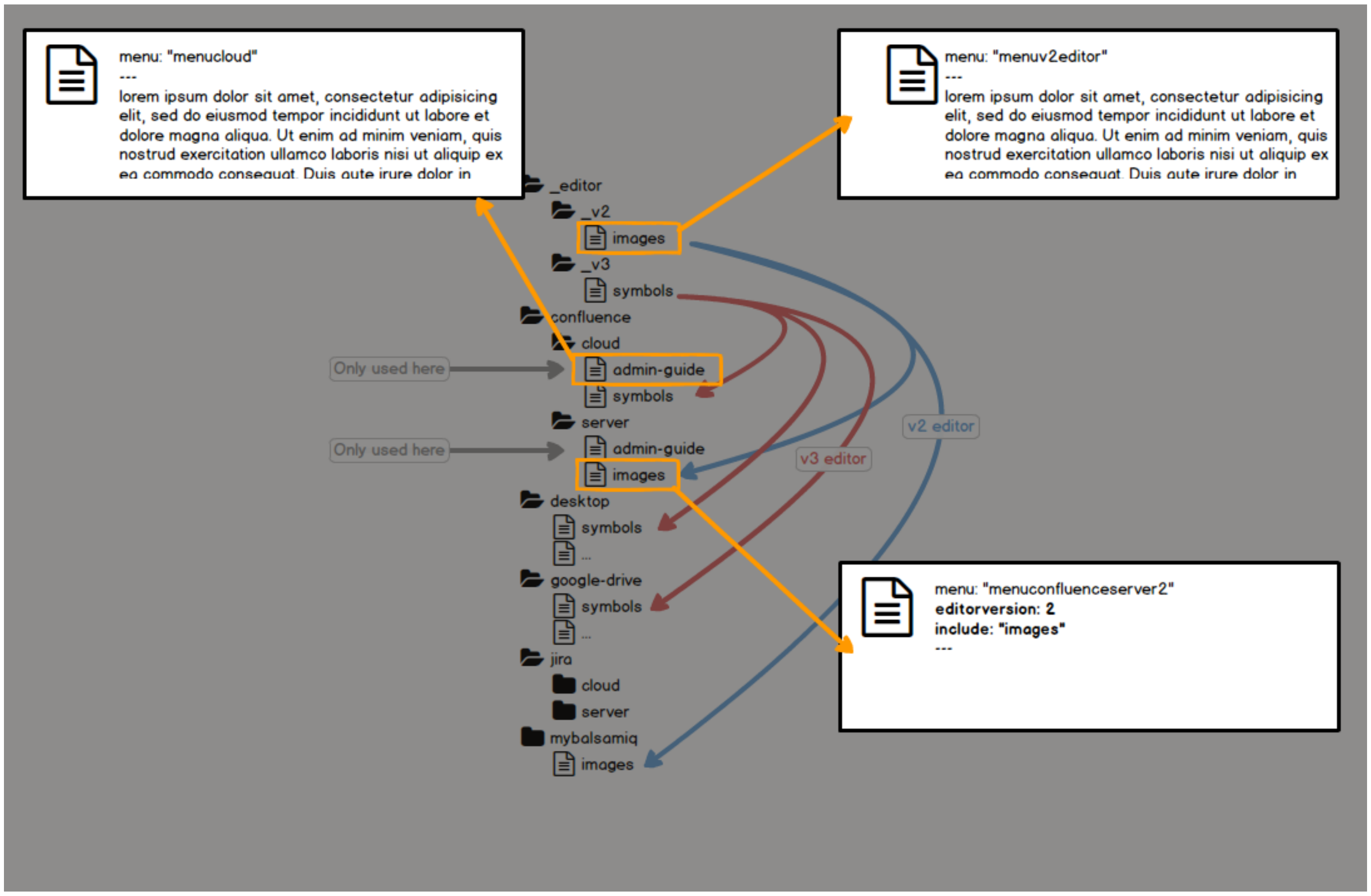
Instead of changing file formats, we leveraged the flexibility of Hugo to use the metadata (front matter) from the Markdown file as a trigger to tell Hugo to insert content from one file to another.

The code looks like this:

```
{{ if .Params.include }}
 {{ $include := .Params.include }}
 {{ if eq .Params.editorversion 3 }}
   {{ range where .Site.Pages "Params.menu" "menub3editor" }}
     {{ $file := .File.BaseFileName }}
     {{ if eq $file $include }}
       {{ .Content }}
     {{ end }}
   {{ end }}
 {{ else if eq .Params.editorversion 2 }}
```

```
   {{ range where .Site.Pages "Params.menu" "menub2editor" }}
     {{ $file := .File.BaseFileName }}
     {{ if eq $file $include }}
       {{ .Content }}
     {{ end }}
   {{ end }}
 {{ else }}
   {{ end }}
{{ else }}
 {{ .Content }}
{{ end }}
```

Hugo looks for two pieces of front matter. The first is the presence of a parameter called `include`. If that exists, it looks to see which version of the editor to use. Then it copies the content from *the file with the same name* from the core documentation folder ("_v2" or "_v3") specified in the editor version parameter.


menu: "menucloud"
---
lorem ipsum dolor sit amet, consectetur adipisicing 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. Duis aute irure dolor in
menu: "menuv2editor"
---
lorem ipsum dolor sit amet, consectetur adipisicing 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. Duis aute irure dolor in
_editor
_v2
images
_v3
symbols
confluence
cloud
admin-guide
symbols
server
admin-guide
images
desktop
symbols
...
google-drive
symbols
...
jira
cloud
server
mybalsamiq
images
Only used here
Only used here
v2 editor
v3 editor
menu: "menuconfluenceserver2"
editorversion: 2
include: "images"

If the `include` parameter isn't present, Hugo just uses the Markdown content in the file without copying content from elsewhere. This allows us to mix and match docs that are unique to one product with docs that share functionality with multiple products.

It took some effort to get it set up, but now we don't even think about it. It's really easy for the content writers, and we'll be able to extend it when we add yet more versions in the future (we already are, in fact).

Previous
← *Case Study - Symantec, Jennifer Rondeau*

Next
Animated GIFs Pause and Play - Balsamiq, Leon Barnard →