Getting started guide

Table of Contents

This is a pilot for an experimental way to develop short-form (less than 100 pages or so) content for O'Reilly. Here's how it works:

- **COMING SOON**. We supply you with a Gollum [https://github.com/github/gollum] instance, the awesome, git-powered wiki from the folks at GitHub [http://www.github.com]. You write/edit your document like you would a wiki, and you can share it with friends, editors, and other people you think might help.
- You write in AsciiDoc, an open source text format developed by Stuart Rackham [http://www.methods.co.nz/asciidoc/] (more on this in a bit). Every time you save a page, it gets logged into a git repository, so it's always under version control. (We can grant you access to your repo — just ask!)
- When you're ready, we suck the AsciiDoc into our publishing toolchain, convert it to DocBook (our native format), and then push it out through various channels. You can read a bit about this at Matt Neuberg's blog [http://www.apeth.net/matt/iosbooktoolchain.html], where he describes how he wrote his "Programming iOS" book.

So, why bother with this, you ask? Mostly, we're trying to create a more appealing process that will get you writing as quickly as possible, not mucking around with a bunch of tools. The cool thing about AsciiDoc, Gollum, and git (and GitHub!) is that they can help ease a lot of the complexities of the publishing process, letting you (hopefully!) focus on the what you want to say.

This document covers:

- Getting started with AsciiDoc
- Working with figures and cross-references (XREFS)
- Putting code examples on GitHub
- Structuring your overall document

- Tips and tricks

1. AsciiDoc Quickstart

AsciiDoc [http://www.methods.co.nz/asciidoc/index.html] is a text document format for writing (among other things) books, ebooks, and documentation. The main advantages of AsciiDoc are that it is easy to use and plays well with O'Reilly's publishing process. It's similar to wiki markup — if you can write a Wikipedia article, then you're pretty much 90% of the way there. This Asciidoc cheat sheet [http://powerman.name/doc/asciidoc] covers a lot of the nitty-gritty, but the following sections will give you an overview of the markup you'll use most frequently.

Note

You can get the complete nitty-gritty on how O'Reilly uses AsciiDoc from our official AsciiDoc guidelines [https://prod.oreilly.com/external/tools/docbook/prod/trunk/samples/r_and_d/asciidoc/]. You'll need to enter "guest" as the username, leave the password blank, and then navigate to the "pdf" directory to download the docs. In addition, the sample chapter with markup [https://prod.oreilly.com/external/tools/docbook/prod/trunk/samples/r_and_d/asciidoc/chapter.asc] is particularly helpful in explaining how things work.

You can create and edit AsciiDoc in any text editor, and then paste it into the wiki to push it into our system. If you're on Windows, you can use Notepad (or anything you want, really). If you're on a Mac, you can use TextEdit, TextMate, or any of a number of choices. The important thing is that you use the AsciiDoc markup.

Here are some links to AsciiDoc books that you can use for reference:

- Codebox: Adventures with Processing [https://github.com/odewahn/codebox3]
- MintDuino Project Book [https://github.com/odewahn/mintduino]
- Best of Radar: Data [https://github.com/odewahn/best_of_radar_data]

1.1. Inline Elements

Here's some some *italic* and some `monospaced` (aka "constant-width" or "CW") text.

Backticks can also be used for literal (CW) text, for example: `ls -al`

Warning

Using inlines in AsciiDoc can be tricky.

Delimiters may not be interpreted as intended if they don't abut whitespace on both sides; the fix for this is to double them up, as explained under "Constrained and Unconstrained Quotes" [http://www.methods.co.nz/asciidoc/userguide.html#X52] in the AsciiDoc User Guide. For example, compare how these render in the PDF: *+foo+ bar* vs. *`foo` bar*

1.2. Hyperlinks

To add a hyperlink, just type the URL followed by the description in brackets. Do not put a space between the brackets and the URL. Here's an example:

```
http://oreilly.com/[O'Reilly Media Website]
http://www.makezine.com/[Makezine]
https://github.com/odewahn/codebox2/blob/master/code/fractal_barnsley.pde[Fractal Exampl
```

1.3. Notes, Warnings, and Sidebars

If you need to add a note or warning, here's the format:

```
[NOTE]
===============================
O'Reilly Animal books traditionally make no distinction between the
DocBook +<note>+, +<tip>+, and +<important>+ elements.
===============================

.Add a Title
[WARNING] .Add a Title
===============================
O'Reilly Animal books traditionally make no distinction between the
DocBook +<warning>+ and +<caution>+ elements.
===============================
```

Here's a Sidebar:

```
.Titled vs. Untitled Blocks
****
O'Reilly house style generally uses titles only on formal blocks
(figures, tables, examples, and sidebars in particular). Although
AsciiDoc supports optional titles on many other blocks, these
generally are not appropriate for O'Reilly books.

Conversely, _omitting_ a title from a sidebar is generally not
conformant to O'Reilly style. If you have questions about whether
content belongs in an sidebar vs. admonition vs. quote or other
element, please consult with your editor.
****
```

1.4. Code

Enclose code samples inside 4 consecutive minus signs ("-"), like this:

```
----
    if (x < X_MIN) {
       X_MIN = x;
    }
    if (x > X_MAX) {
      X_MAX = x;
    }
    if (y < Y_MIN) {
      Y_MIN = y;
    }
    if (y > Y_MAX) {
      Y_MAX = y;
    }
----
```

You can also use the AsciiDoc "include" macro to pull in code files (more on this in ??? section):

```
----
```

```
include::code/example.c[]
----
```

1.5. Bullet lists

Use asterisks ("*") to create bullets. ou can indent items by using multiple asterisks:

```
* Lorem ipsum dolor sit amet, consectetur adipiscing elit.
* Nulla blandit eros eget velit bibendum placerat.
** Pellentesque id justo ultrices est pharetra suscipit.
** Cras nec magna a lectus consequat varius.
* Phasellus tempor lacinia neque, et scelerisque lectus luctus id.
```

The list will look like this:

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Nulla blandit eros eget velit bibendum placerat.
  - Pellentesque id justo ultrices est pharetra suscipit.
  - Cras nec magna a lectus consequat varius.
- Phasellus tempor lacinia neque, et scelerisque lectus luctus id.

1.6. Numbered lists

Use periods (".") to create a ordered (i.e., "1, 2, 3, …") lists:

```
. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
.. Nulla blandit eros eget velit bibendum placerat.
.. Pellentesque id justo ultrices est pharetra suscipit.
. Cras nec magna a lectus consequat varius.
. Phasellus tempor lacinia neque, et scelerisque lectus luctus id.
```

Here’s how it will look:

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   a. Nulla blandit eros eget velit bibendum placerat.
   b. Pellentesque id justo ultrices est pharetra suscipit.
2. Cras nec magna a lectus consequat varius.
3. Phasellus tempor lacinia neque, et scelerisque lectus luctus id.

1.7. Simple Tables

Here’s the basic format for creating tables:

```
.An example table
[width="40%",options="header"]
|=============
|col 1| col 2| col3
|1  | 2 | 3
```

```
|4  | 5 | 6
|7  | 8  | 9
|=============
```

It will look like this:

Table 1. An example table

col 1	col 2	col3
1	2	3
4	5	6
7	8	9

2. Figures and cross-references (XREFS)

Do not use the JPEG file format for saving screenshots, as they work poorly in print. Please use GIF, TIF, PNG, or BMP. To add a figure to your project, just place the figure file in the "attachments" directory and add a link, like this:

```
[[figure1]]
.Put the caption here
```

```
image::attachments/figure1.png[scaledwidth=90%]
```

Figures can "float" throughout the document and may not be on the same page you expect them to be. For this reason, saying things like "This figure…" or "The figure below…" might confuse the reader, since the figure may very well be on a totally separate page once the document is rendered. For this reason, all figures must have an accompanying cross-reference.

Cross-references (XREFS) allow you to link to other sections in the document, and are most commonly used to refer to figures. XREFS have two elements: an anchor, which is what you're linking *to*, and a reference, which is where you're linking *from* in your text. Anchor links are denoted by double brackets. Reference links are denoted with double angle-brackets.

Here's a sample of how you'd reference a figure:

```
You can see an example of foo in <<foo-figure>>.  Notice how beautiful the foo is.  Mmmmm
```

```
[[foo-figure]]
.Here is the caption for foo
```

```
image::attachments/some_figure_related_to_foo.png[scaledwidth=90%]
```

To generate a cross-reference, use this syntax:

```
<<ID>>
```

where `ID` is the anchor or BlockID of the target, which you place in double square-brackets above that block.

3. Code samples

We recommend that you place all code samples on GitHub. This makes it easy for you, and more importantly, your readers, to pull down the code and fix it. Take a look at Matthew Russell's Mining the Social Web [https://github.com/ptwobrussell/Mining-the-Social-Web] to see how well this works when done consistently.

Note

If you don’t want to put your code on GitHub, you can simply place the listing files in the "attachments" directory and then use the "include" macro to pull it in, like this:

```
----
include::attachments/my_program.py[]
----
```

Matthew’s included a lot of niceties on the landing page, like marketing text, praise quotes, JPEG of the front cover, a plea not to steal the book, etc. We really like the way he’s set up the main page, but we’re certainly not mandating authors include all this material, or that they adhere to any formal strictures. If you prefer to do something much more bare-bones, that’s totally fine, too. See the Jonathan LeBlanc’s Programming Social Applications [https://github.com/jcleblanc/programming-social-applications] for something far more simple.

Note that Jonathan’s actually organized his code in directories by chapter, which we think is a good idea if there are a lot of code examples and/or chapters in the book. It makes it easier for readers to find things. Matthew Russell’s solution for enhancing findability/navigability was to add an Examples Listing [https://github.com/ptwobrussell/Mining-the-Social-Web/wiki/Numbered-examples] to his GitHub wiki.

In terms of linking to GitHub content from the book, we’d recommend you do the following:

3.1. Add preface text

Add the following text to the "Using Code Examples" section of the Preface:

```
The code examples in the following chapters are available for download at GitHub at
https://github.com/<username>/<booktitle>/ -- the official code repository for this
book. You are encouraged to  monitor this repository for the latest bug-fixed code as
well as extended examples by the author and the rest of
the social coding community."
```

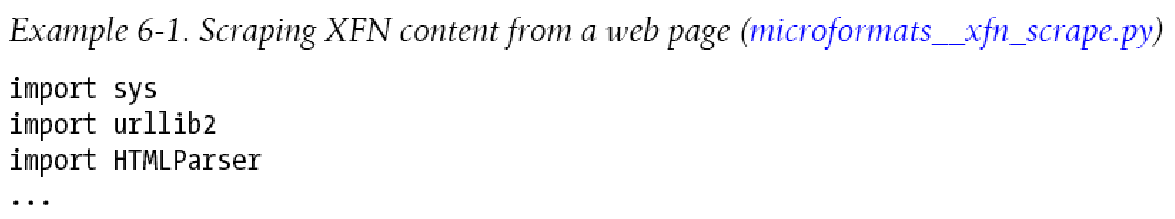
3.2. Link out to your examples in the text

For every code listing that’s in both GitHub and the book, add a hyperlink in the book to the corresponding code in GitHub. If the code is in a formal example, we recommend putting the link in the Example title (otherwise, you may want to add the link to the preceding body text). Here’s an example of the AsciiDoc markup you can use (note the backslash after "microformats" in the URL, which is necessary to escape the double underscore so it is properly translated to PDF):

```
[[EXTEST]]
.Scraping XFN content from a web page
 (https://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/microfor
====
----
import sys
import urllib2
import HTMLParser
...
----
====
```

Note that for the hyperlink node text, we’ve used just the filename of the code, so that the full URL isn’t displayed in the Web PDF. The elemnts will appear as in Figure 1, “Here’s how references to your code on GitHub will appear in text.”.

Figure 1. Here’s how references to your code on GitHub will appear in text.



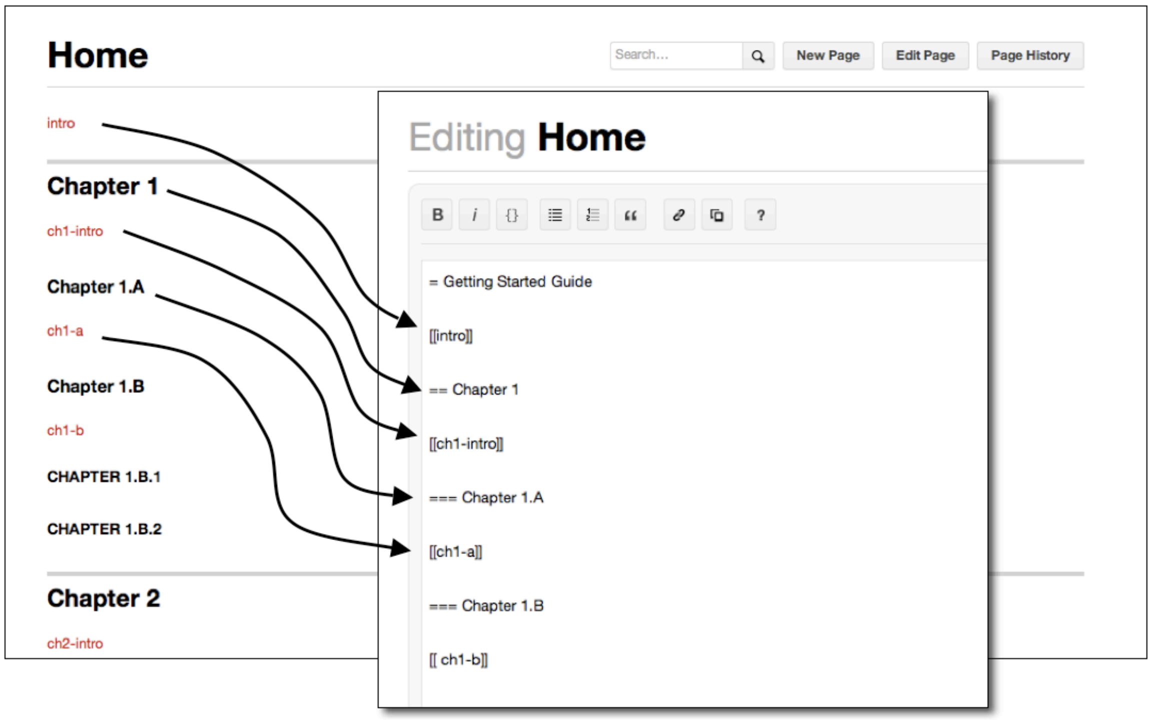
Example 6-1. Scraping XFN content from a web page (microformats__xfn_scrape.py)

```
import sys
import urllib2
import HTMLParser
...
```

4. Structuring the document

Creating more complex documents is simply a matter of stitching together smaller sub documents. While there are a variety of approaches, the one we’re recommending right now (this is a pilot, after all, so this might change) is to use the special file "home.asciidoc" as a granular TOC or index page. So, basically, you put all the section headers that describe the piece’s structure, and then have links out to the content pages. Keeping the structure in one place (home) will make it easier to move things around as you rewrite and edit. The idea looks something like Figure 2, “Structure the overall document using headers and links”.

Figure 2. Structure the overall document using headers and links



Home

intro

Chapter 1

ch1-intro

Chapter 1.A

ch1-a

Chapter 1.B

ch1-b

CHAPTER 1.B.1

CHAPTER 1.B.2

Chapter 2

ch2-intro

Editing **Home**

```
= Getting Started Guide

[[intro]]

== Chapter 1

[[ch1-intro]]

=== Chapter 1.A

[[ch1-a]]

=== Chapter 1.B

[[ ch1-b]]
```

So, how do you do this? Basically, just put all your structural headings (defined by nesting equal signs at various depths) in the home file, and then put internal links (which are the page name enclosed in double brackets). Note that you don’t have to create each content page first — you can simply create a link the link. When you click on it, the new page will be created automatically. The following sample should give you the basic idea.

```
= Getting Started Guide

[[intro]]

== Chapter 1

[[ch1-intro]]

=== Chapter 1.A

[[ch1-a]]

=== Chapter 1.B

[[ ch1-b]]

==== Chapter 1.B.1

[[c1-1-b-1]]

==== Chapter 1.B.2

[[c1-1-b-2]]

== Chapter 2

[[ch2-intro]]

=== Chapter 2.A

[[ch2-a]]

=== Chapter 2.B

[[ch2-b]]
```

In addition to making the document more manageable, this approach also has the benefit of helping you think through the outline and structure of the document.

5. Tips and Tricks

- Write in a text editor and paste the content into the Gollum wiki
- Don’t put section headers inside your content sections — put them in the "Home" file
- Don’t use inline formatting
- Don’t use footnotes
- Don’t have an empty section

- Don’t start an xref with a number or character
- Don’t duplicate an xref name
- To generate a PDF from this repo, use this command: "a2x -fpdf --fop --no-xmllint README.asciidoc"