# *Menus in AsciiDoc*

*by Tom Swan*

When it comes to writing notes for my music and computer projects, I'm like a chef prepping a restaurant menu. I've got so many different pans on the stove going at once I need good notes to keep track of my concoctions and their ingredients.

I'm used to a bit of chaos, but one day several years back I realized that computers were not helping me to use my notes effectively. In fact, computers were making the situation worse. Taking notes ain't the problem — *finding* them in my convoluted file systems makes needles in hay stacks by comparison look like telephone poles.

I had notes in Page format on my Mac systems, Word files in Windows, Open-Office notes on my Linux workstations, plain text files on all of those, plus paper notebooks and pads stuffed in drawers, stacked in corners, heaped in boxes and stained with coffee, tea, wine, and here and there ornamented with dried-up splotches of either chocolate or spaghetti sauce, I can never tell. I could probably eat my notes and survive.

You think I exaggerate? When I moved out of my college dorm room, my friends used a *shovel* to load my trailer! I am *not* making that up!

Top of Page

## Scripting to the Rescue

There's probably no cure for my messy paper addiction, but I *have* found the perfect solution for compacting my digital trash: *Asciidoctor*.

> *AsciiDoc* is a simple but powerful scripting language for plain-text writing. *Asciidoctor* is a program that converts AsciiDoc documents from text to HTML and PDF formats for viewing on- and off-line, for printing, for assembling books, for writing blog postings like this one and for creating many other types of documents. I write with Asciidoctor almost exclusively now — in fact, I used it not only for these words, but also to create almost my entire web site tomswan.com!

Those who have tried AsciiDoc (the language) and Asciidoctor (the program) soon realize one of this tool's key strengths. It's unequaled for turning plain text into presentable web pages. It lets me write in plain text — think *manuscript* — but easily get great-looking output. It's not the perfect solution for my problem, however, because it does nothing to help me *use* those results in creative ways.

That's not a criticism of AsciiDoc. I love it, and Asciidoctor is an amazing (and blazingly fast) text processor. It's open-sourced (no charge!) and is meticulously maintained by some very caring souls. You can find it at: http://www.asciidoctor.org.

But as useful as it is to me, AsciiDoc is not a web-design language nor is it a publishing system. It's a *scripting language*. You feed it text and out comes (usually) a perfectly formatted HTML or PDF file, ready for publishing. Other formats are certainly possible, but those two are the ones I need most often.

What I lacked, I soon realized, was a system powered by Asciidoctor but capable of organizing and accessing compiled HTML and PDF documents in a *fully-developed web site*.

I am still developing that system (it's well along and I'll be writing more about it here soon). One of my system's key elements, which I'll describe in the rest of this article, is how it adds navigation menus to compiled AsciiDoc documents in order to create a web site of my notes that is fully searchable, cross-referenced, and linked, whether for private keeping in a hard-drive folder, published on my local intranet, or launched into the internet cloud for the entire world's consumption.

One of the key elements in my system is a navigation menu. Here's how I create it using only stock AsciiDoc features.

Top of Page

## What's on the Menu?

What is a menu but a list of links, and what is a menu navigation bar but a table of such links to a site's urls? In a site with About, Videos, Blog, and Photos pages, to create a navigation menu for going to those places, all we need is a table of links to the HTML file urls.

In AsciiDoc, links like that are created as *macros*, that is, symbolic names that stand for something else. Here are some macros I might use:

```
:home: index.html[Home]
:page1: page1.html[Page-1]
:readme: README.html[Read-Me]
```

Type all code lines flush left in the source document. They are indented here just to distinguish them from the text.

The `home` macro encased in colons refers to the site's main `index.html` file, opened by the server when the home page's location (`www.yoursite.com` for example) is accessed. The text in brackets is what's shown in the text (empty brackets show the url itself), so if I write:

```
Click link:{home} for the site's main page
```

the HTML result in a browser comes out like this: Click Home for the site's main page. When enclosed in curly braces, macros are *expanded* to the text they represent. The word `link:` with its colon tells AsciiDoc to treat that text as a selectable link.

Notice how readable the plain text and browser formats are. They are nearly the same.

Browse to Source Text to see the text for this post *exactly* as I wrote it. (This works only online.) You can read the article and understand it in either form. Try doing that with HTML code — I did, once. HTML is brilliant, no question, but I don't want to be reading and writing prose in it.

With AsciiDoc, your original text remains highly readable, making writing more enjoyable while still retaining a great deal of control over output formatting. There are no opening and closing tags as there are in HTML — and please don't even *think* about manually using XML. AsciiDoc source text is nearly as readable as its processed results. Very cool.

Top of Page

# Downloading and Trying the Demo

One way to create a menu of macros, and in that way provide a clear method for traveling to and from a web site's pages, is to construct an AsciiDoc table containing as many links as you need. The Figure shows such a menu in action.

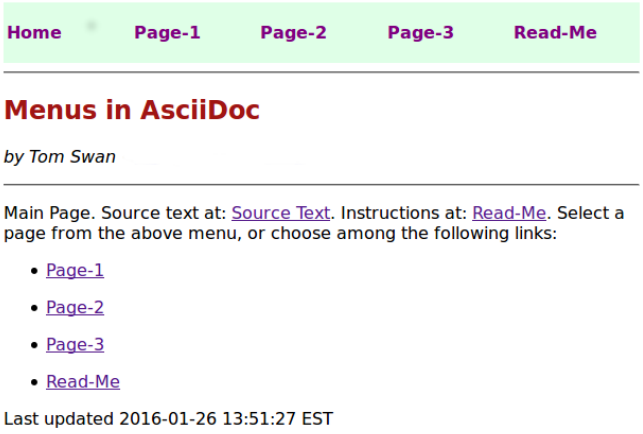


*Figure 1. Figure. MIA (Menus in AsciiDoc) Demo*

In the text, declare the menu as a table of links:

```
[.main-menu]
|===
|link:{home}|link:{page1}|link:{page2}|link:{page3}|link:{readme}
|===
```

Tables in AsciiDoc begin and end with a vertical bar followed by three or more equal signs. Inside the table, each link is in the usual form. The vertical bars represent cell boundaries — they aren't displayed. The `.main-menu` declaration in brackets denotes a class name for this element. Look in the demonstration's `menus.css` (stylesheet) file and you'll see how this name is used for selecting the menu's font, color, and other display options.

You can try out my menu system now by selecting this link (use browser navigation to return here):

- Menus in AsciiDoc — A Demonstration

To download a ZIP file with all of the demonstration's source files, select this link (or right-click it and select `Save Link As…`):

- Download Demonstration Sources ZIP File

Unpack the ZIP file and examine the resulting `mia` folder in the current path. Go to that folder and, if you want to rebuild the demo, from a command line, type `make`. Open the resulting `index.html` file to view the demonstration site in a web browser.

For those who aren't command-line Linux fans like me, just run the demo by opening `index.html` and load the sources if you want into your favorite web designer.

Top of Page

# Understanding the Code

The demonstration is straightforward and the source files shouldn't be hard to understand. Read the comments in the code and also see the `README.txt` file in the download.

Files ending in `.adoc` and `.txt` are plain ASCII text files. I like to keep declarations and macros and such (call them "settings") in `.adoc` files, and the document's actual text in `.txt` files. You don't have to do the same, but when I'm writing prose, I don't care to stare at a bunch of odd looking symbols — I just want to see my text. So I keep things separate. My quirk I guess.

Central to the design of my system is the AsciiDoc `include` command. Use it like this:

```
include::menu-include.adoc[]
```

Assuming Asciidoctor can find the named file, it reads and processes the file's text before moving on. (You probably know that most programming languages have a similar command. The `make` utility calls it `source`, though. Go figure.)

You can add a path name if necessary:

```
include::_includes/menu-include.adoc[]
```

To process or *compile* a source text file, at a command line prompt, type:

```
asciidoctor myfile.txt
```

The result is `myfile.html` in the current folder. Open that to see your text in a browser. Go ahead and try it now. You don't need even headers or any declarations at all, just plain text, whatever you want to type. Save your text in a file *as plain text* and compile it. And then open the resulting `myfile.html` to see your writing published in a web browser.

The demo consists of multiple .adoc, .txt, and other source files. The resultant .html files are most easily created using `make`. Otherwise, you have to compile each file individually for the demo to work.

Top of Page

# Thanks for Reading!

If you are an AsciiDoc fan like me, try my menu system in a project and let me know how that goes. Good luck!

***Writers!***

If you are new to AsciiDoc (remember, *Asciidoctor* is the program), I hope you will give it a try. Writing in plain text might be more natural for me because I learned my evil ways by banging on a Smith Corona and going through reams of cheap newsprint since that's what I could afford. I wrote my early books that way. But even if it's not natural for you, I urge you to give writing in plain text like this a try — especially if you are still wasting time positioning margins and dragging whatnot around. Maybe it's just me and my ways, but I'm a writer. I don't want to be a typesetter. I'd rather let the computer make my writing look good so that I can have more time for other things!

---

Now that I have navigation menus available as I attempt to round up my scattered collections of text files from all over creation, at least the notes I've upgraded so far have never looked better, and my printouts taste just as great. Okay, not really. I was only kidding about the spaghetti sauce.

It's probably strawberry jam.

Top of Page

©2017 by Tom Swan * www.tomswan.com * 2.3.6 * Web Writer 1.0.1