Asciidoc Writer's Guide

Table of Contents

Intro

Let's start writing!

Before you start writing, some general thoughts make sense to clarify what Speaking, Reading and Writing are all about. What are the differences, what the difficulties. I recomment to groove into that topic to become acquainted with the base how human use a language rather than step in directly to tools and techniques for writing.

Speaking is Natural

Spoken language is hard-wired inside the human brain. Language capacity in humans evolved about 100.000 years ago, and the human brain is fully adapted for language processing. Any healthy child will learn to talk because it is an innateness.

Reading and Writing Are Not?

A related fact should be self-evident: Reading and writing are acquired skills for which the human brain is not yet fully evolved. Human brains are naturally wired to speak. They are not naturally wired to read and write. With teaching, children typically learn to read at about age 5 or 6 and need several years to master the skill. Sophisticated reading comprehension is the goal of 8 to 16 more years of schooling. As all of you know, a long time, a long way to go.

Writing is Unnatural

Writing is an unnatural activity. Unlike talking, which all people learn on their own in the process of growing up, writing needs to be taught and requires the use of tools.

You can carry on a conversation at any time and any place. You don't need any special equipment. The medium of speech is the air we breathe, and speech itself is just a different way of breathing. You do the whole job with just your own body. For that reason: Speaking is natural.

But in order to write, you need specialized tools. You need a tool to write with!Ñ!a pencil. You need something to write on!Ñ!you need paper. But the most important tool we use in writing is one we often don't think of as a tool at all. You need a set of recognizable symbols to represent words!Ñ!an alphabet.

The alphabet is a set of tools. Unlike the words we speak, the letters of the

alphabet had to be invented by people. You learned to speak your native language by just hanging around and hearing it. But when you learned to read and write you had to sit down and study!Ñ!first to learn to recognize the letters and associate them with the correct sounds and then to produce them yourself with pencil and paper. That's basically the reason why it is hard to write.

Speed

Stop for a moment and think about the differences between writing and talking, because those differences account for much that is difficult!Ñ!or appears to be difficult!Ñ!about writing.

First, writing is much slower than talking. It will take several times longer to type the sentence that makes up the previous paragraph as it takes to speak. Try the experiment yourself. Even with the wonderful tools available to us today, writing takes more time. Sometimes much time.

Why is that important? One reason is that our brains are conditioned to work at the speed of talk, to react experientially to changing circumstances and to give feedback very quickly. Writing makes us spread out the process of expressing ourselves. That can be very uncomfortable, and it can require us to learn a whole new way of experiencing language.

Loneliness

Second, we always!Ñ!or almost always!Ñ!talk to other people. But we write, in an important sense alone. When I speak or listen I am usually looking at someone else, but when I write I am looking at myself or at what I am writing. Most of us learned to talk by looking at other people talk while we listened to them talk. And to this day, when we speak to someone we are usually looking at the person we are speaking to.

Sight and Hearing

We have learned to manage listening without seeing the speaker!Ñ!we can talk on the telephone or listen to the radio.

But even then, we can imagine the expressions that accompany the speaker's voice. Why is television more popular than radio? Because it is more involving and natural to be able to see the person we are listening to. When we speak or listen, we usually have direct input from another person through two senses: sight and hearing.

And it is dynamic, rapidly changing input, at the speed of natural speech. When

we read or write we are not looking at another person at all and we are not hearing anything that's relevant to the meaning of the words. In reading, if we can read fairly rapidly, we can imagine the speaker speaking; we can hear in our minds what we can't hear in our ears. But when we write we have to watch ourselves writing.

Because writing requires using complex tools (pen, paper, alphabet), we must look at ourselves doing it in order to get it right. Try this experiment: write longhand at a normal speed with your eyes closed or holding a piece of paper over your hand to hide the paper you're writing on. Many people cannot write at a normal speed under these circumstances. Those who can often produce a childish-looking scrawl that is hardly legible.

Written language is visible

If you are a touch typist, of course, you don't need to look at the keyboard as you type, but you do need to constantly monitor what you are typing on the screen or page. The fact that you do not get input from other people while you are writing means that writing requires a much higher degree of concentration than talking, listening, or even reading.

That is both good news and bad news. The bad news is that writing can seem a lonely business, one in which you have to wait a long time to get the feedback that would come instantly if you were talking to someone.

Written language is permanent

The third difference between writing and speaking is that writing is permanent, and therefore more public, than spoken words. When you speak, as soon as the words are out of your mouth, they are gone. They exist only in the mind of the hearer. So speech, being so impermanent, is very changeable.

If you see the person you talk to doesn't catch what you mean the first time, you'll just say it again, maybe a little different. If the person you talking to looks confused, you'll explain what you've meant. If you've ever read a direct transcript of someone speaking, you might have noticed how many fragments and false starts are commonplace in spoken language. We don't notice this very often as listeners because we are so used to it. As listeners and as speakers, we automatically erase errors and misstatements: a brain is an excellent filtering engine and replaces what does not make sense.

When we speak, we are constantly revising what we are saying. It's easy to do. When we write, on the other hand, there is a record of what we've written that never changes and that is the same for all readers. Anyone who gets ahold of

what we've written will see exactly the same thing. This permanence is one of the great strengths of writing. It preserves our ideas and allows us to spread them to as many people as can read them.

Differences of Speaking and Writing

So there are at least three important differences between speaking and writing. Speaking is faster; writing takes more time. Speaking is social, done in the presence of other people.

Writing is in a sense private, requiring us to focus on the act of writing itself. Speaking is transitory, writing is permanent. We should think about these differences as we try to understand and grapple with the task of becoming better writers.

One conclusion we can draw is that writing is, in some ways, inherently difficult. That is to say that the difficulty is caused by the very nature of the act of writing, so we will always encounter that difficulty. Writing, in this sense, is not like walking or talking or driving a car. These activities become easy with enough practice. Writing: never.

Writing needs continuous practice

Writing is more like sports or making music. It will always remain difficult even after you have done it a lot. In the sense that just doing it well takes much effort and concentration. This is bad news!Ñ!and good news.

The bad news is that if we are expecting to get to some point where we can pass some test or class, master some technique or gimmick, or reach some level of practice where writing will no longer require real effort, then we are bound to be disappointed.

The good news is that almost all the tasks that we seek out for their own sake in life are difficult in this way. No one devotes her life to casual walking. But sports, playing the guitar or writing can become a career or even an obsession. In the long run, only inherently difficult tasks are interesting and involving.

If you find writing difficult and feel that you shouldn't or that you are alone or defective in having this experience, you can relax. It's difficult for you; it's difficult for me; it's difficult for all people.

Meet the Challenge

People who want to master any complex skill have a similar experience. If what you are looking for is a way to get results without any effort, you are wasting your time. But there are ways you can make your efforts count for more. Some of those things are suggested by the characteristics of writing that has been discussed.

Since talking, listening, and reading are all easier than writing, you should use them to prepare for writing. It is much harder to decide how to say something before you have said it. And it is definitely harder to decide how to say something in writing that you have never said in conversation.

Talk to people about what you believe. Test your ideas in the faster, less permanent medium of speech before you try to set them down in the slower, more permanent medium of writing. Read all you can about what you want to write about, and then talk to someone about it. Remember that you will have no chance to see how people react when you are writing to them, but you do have a chance to see how they react when you are talking to them.

Use the skills you have to help you to develop new ones. It is easier for all of us to talk than to write. So even when you are writing, talk it out. Try to say what you think you want to say out loud, then write it down.

As you write, stop to read what you have written aloud, so you can hear it. Talking and hearing bring parts of your brain into the task that will help you to focus on what you are saying and achieve the level of concentration you need. If you find your concentration wandering while writing, read what you have already written aloud, and then imagine you are talking with someone about it and carry on an imaginary conversation. Talking will help to ease you into writing in a more concentrated way. As far as I can tell, very experienced writers have developed the ability to hear what they write, so that for them the act of writing is closer to the act of speech than it is for the rest of us.

Time counts

Assume that writing will always take longer than you expect it to. Having an idea and writing the idea are different processes. Writing takes longer than talking. And because we lack the feedback when writing that keeps moving us along when talking, we are going to stop and start much more when writing than we would if we were explaining our ideas to someone. And because writing is permanent, we are often afraid to put our ideas down on paper.

All of these things, which are normal and inevitable, mean that if we try to force

ourselves to write fast, we probably won't write at all. If you think that the only way you can write is under a deadline and so put off writing until the last minute, you do yourself a great disservice. You have no idea what you could do if you gave yourself the time to do it.

On the other hand, while we cannot force ourselves to write fast, if we can let ourselves write fast we will find that writing seems smoother and easier and more natural. The difference here is the difference between force and let. When we are thinking most experientially, when we are getting ideas out in the open most effectively, is when we experience writing as most like talking. It won't be as fast as talking, but it may seem even faster when we are doing it.

What slows us down is that we are all afraid of the permanence of writing. We are afraid of making mistakes, of saying something that we didn't really mean, of appearing foolish. We can gain some control over this fear if we realize that we have some control over how permanent and how public our writing really is.

We can keep some of our writing fairly private, letting people we trust look at it and give us feedback. We can test our writing before we make it public. If I know that the first draft of my essay will be seen only by me or only by a few people, I can relax and say whatever I want. I can let myself write my first draft fast because it is private. I can always fix it later.

Writing is Natural

Because writing is private when we produce it, we can sometimes believe that it can remain private. Much of the writing we do in school seems this way. We write only for the teacher, and we trust the teacher never to breathe a word. In fact, much of the student writing that I have seen as a teacher was apparently written by people who didn't expect anybody else to ever read it, or at least pay any attention to it. Writing that is just a classroom exercise, that has no real audience, is a waste of everyone's time.

Why?

Well, ask yourself this: What good is writing, potentially, to you? The answer has to be, in some form, that writing is a way you can influence people: change their behavior, open their minds, bring them around to your way of thinking, make them happy, make them angry, get them to ask questions, get them to answer questions. Writing will be important to you in your life because it is a way for you to shape your world. But it can do that only if people read and understand it. It will never make much difference to you whether you can produce a well formed sentence, unless the people who you want to influence read and understand that sentence.

What was said above about writing being unnatural is true in a way, but in a deeper sense writing is completely natural. It's different than speaking, but underneath all of the differences, we write for the same reason we speak: we want others to understand us and we want to understand them.

Writing is improved speech

If we emphasize how different writing is so much that we miss its fundamental purpose, then we miss the whole point. Writing is improved speech. It is harder because it is better. It takes longer because it can carry more meaning.

It requires more concentration because it carries more weight. It is more permanent because it is tested and refined. Writing is harder than talking because it counts for more, not less.

What to do next

As said in the beginning: Let's start writing!

But a different way!Ñ!using new tools and new techniques for easier (not easy!) writing and better results. The following chapters help you manage the act of writing better. Simple tricks to get you into a flow of writing, modern but simple tools for text processing with no need of horrible computer programs like word processors.

You learn non disturbing techniques for better concentration on what you want to express, you want to write down. You'll find better ways to structure your writing for better structured, better understandable results.

Sounds good? Go ahead!

Start typing

- Explain the nature of the Act of Writing
- Introduce formal, mindless steps on the way to a chapter, section, books
  - document skeleton, the base structure
  - first step: the top-level topic
  - practise the use of an editor

At eos veniam non vel quaerat voluptates non excepturi quam ut suscipit ea. Dolores expedita voluptatibus autem facere corrupti esse sint vel iusto. Eum voluptatum laudantium assumenda earum beatae libero suscipit maxime quis architecto ipsa. Consequatur quia qui possimus veniam sequi qui corrupti dolor. Facilis magni consequuntur quae est sit explicabo beatae minus esse numquam aut iste.

Asperiores eaque quos velit quo voluptatem totam minima sequi et voluptas in ipsum facilis. Vero in reiciendis sapiente occaecati.

Heaven and Hell of writing

Many people find the idea of writing a book or an article difficult. I think this is mainly due to a poor definition of what writing is.

In movies, there is this romantic idea of the process, the act of writing. The author struggles, drama ensues. Then all of a sudden inspiration strikes. The author sits down all the night just listening to what inspiration says. In the morning, among a pile of cigarette butts and empty wine bottles, there is a brand new spanking chef-dﾟ Ï uvre of a manuscript.

Everyone who reads it says: itﾟs the best thing ever! Or, if the author is ahead of his time (as may happen in drama) average Joe is saying: what a shit. But more in a theatrically way.

The author is done. Letﾟs celebrate, while waiting for the next visit by her majesty, the Muse.

Thatﾟs definitely not how it works with sophisticated text, articles or even books. I donﾟt have first-hand experience with fiction, but from what Iﾟve read from fiction writers, thatﾟs not how it works with fiction either. Writing is a scilled craft. A technique, or a bunch of them.

Letﾟs dive into scilled crafted writing.

The White page

When people buy into the romantic idea of the Muse and one-off writing sessions, and they try to write, no wonder they fail. Imagine out hero saying: "Heck! Iﾟm going to write a computer book! I know a lot about Web Design. What a hell should be easier to do as Iﾟm a specialist for the Web!

He creates a new folder, a new file called book.doc and proceeds typing the title: Responsive Design with HTML5 and CSS3. He likes it. Next, a subtitle: Unleash the power of mobile, social, local, yadda and dadda.

OK, so how big should the title be? What about the subtitle? 20 minutes struggling with formatting options in a word processor. The future author goes: Meh, Iﾟll worry about formatting later.

New page. Next White page É Big white page.

The new author let the mind wander for once and perhaps even think about treating himself to a refreshing footbath. The energy will go up from the foots

into the brain to the big white page.

"History of HTML? Mmmmh. Boring. Not that."

Still White Page É

"Should I open with a joke? But what if it falls flat?"

White page.

The next brilliant idea comes on his mind: "Hey man, I'm connected to the internet". Open the browser. Search for writer writing block. The search engine is telling him: 85.400.000 matches. Oh Mama, that's a lot.

Trying the first results. The time goes by. The temperature of the refreshing footbath down.

You see how the prospect of writing is scary for many people.

What can be done about it?

There are some magic words in the wild for making something entirely new: prepare, plan, act and check. Hard to find on a writer's blog and 85.400.000 matches are not that helpful.

Preparation

: You already did this part by having this tutorial at your fingertips. Nothing more needed.

Planning

: It is always a good idea to divide, what you are intended to do, into steps. First unordered, writing down what do you think what's needed. If the a list of top-level items is in place, order them.

Acting

: Go for the first topic of the plan, thhe ordered items you've found.

Checking

: Check the results of what you've done so far on a topic. You'll find some things missing in the preparation or the planning. Add the missing parts and start from the beginning.

The 4 steps are done in a cycle. And this cycle is well known: the Deming Cycle.

Figure 1. Deming Cycle or PDCA Cycle

The Deming Cycle, or PDCA Cycle, is a continuous quality improvement model consisting out of a logical sequence of four repetitive steps for continuous improvement and learning: Plan, Do, Study (Check) and Act.

As mentioned in the intro part: Writing is improved speech. And it is done in cycles. The Deming Cycle is the base for continuous improvement, the process needed for writing, for written language, for improved speech. That means vice versa it is not needed to have all things in place from the beginning. It is an iterative process to complete a step, a task, a level of learning. Furthermore it not recommended to bring all things in place from the beginning, to learn a bunch of things upfront - it won't work.

Back again to the intro part: Writing is more like sports or making music. It will always remain difficult even after you have done it a lot. Playing a violine is a life-long learning process even you're part of a world class orchestra.

For that reason: Do writing. Shake off the bad feelings if something is difficult or went wrong. Nothing you do will appear totally foolish. You are able to do things you had to learn for years. Nobody is perfect. Simple like this: it could be better. It is the next turn on the Deming Cycle, haha!

Be ensured: all of this is part of the game accepted by all serious authors on that planet. Be cool, have fun.

Non disturbing Techniques

As said, writing is in a sense private, requiring us to focus on the act of writing itself. All what is private to humans is embossed by their individuality. No wonder: the act of writing depends on the author. But many things are in common for all writers: avoid all what is disturbing their writing processes, support them by all what is helpful.

A bold statement: If you want to use a computer for writing, never use word processors.

I hear you saying: hey, word processors are for writing. They are used on PC for ages, decades the main application for computers for business. I've never heard word processors playing music!

Yes, that's true. Word processors are great tools for formal text, short one or two pages of text like cover letters. In general: a excellent choice for written business communication up to now. What they are indended for. But a horrible tool for intellectual work.

You'll experience why. You learn that AsciiDoc is, as it's plain text, the better solution for writing demanding text for books, articles or blogs. And, AsciiDoc is both: your creative work and the base for processing engines to transform it into wonderful looking, state-of-the-art documents for the digital world. And last but not least!Ñ!if wanted!Ñ!the base for printing on paper as well.

All the people working in the IT business will agree on that. More than this. They are delighted to hear that writing in AsciiDoc does not need any of the (quite common) word processing tools. What is needed is a good editor - nothing more. Writing in AsciiDoc is done the same way as programmers do for developing since ages up to now - each and every day.

If you're using an editor, an IDE already for your work, go for the toolset you already using. People haven't used editors before, or seldom, the chapter Editor - The Swiss Army Knife help for the first steps.

Divide and Conquer

In computer science, divide and conquer is a design paradigm based on recursion (doing things in cycles). A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the

original problem.

Writers are not nessesarily computer scientists but authors face quite the same challenges: break down complexity to focus on parts simple enough to be managed directly.

Now I come to the helpful practices to make a writer's life easier: what are design paradigm for programmers (see them as writers as well) are split-down techniques like the following.

You can split the process of writing in three phases.

Thinking
: Giving anwers: What am I going to say?

Typing
: Create your content. What ever it is.

Editing
: Improve, fix what you typed. Make the content more readable.

You'll see later on, the process can be transformed into a Deming Cycle for writing. And more than this: the processes you'll learn get fit together. Beside excellent tools, you will need these techniques as writing is an intellectual but handcrafted work.

Step #1: Thinking

Most often people don't think about #1 as writing, But it is. Thinking happens all the time. When you walk, bike, do the dishes, and most often in the shower. Humans cannot prevent thinking without loosing their brains. That's why the aspiring author should make time for activities that seems mindless.

There is a clear deliverable from this process: the TOC (table of contents). Don't see a TOC what it is in the very end. It is important from thhe beginning!Ñ!bullet points of what you want to say. Probably split into chapters, although not until later in the process. The first part is just a list. A todo. You can use paper and pen, a old fashioned but proven way. Or type in your mobile if you don't have paper handy (you're on a walk). This is not as intimidating as the big bad Writing, because the outcome is estimated not more than two to three pages.

Although the more, the better, because it means you have a detailed todo and don't leave much to think about in the next phase.

You can also take this TOC and do a slideshow. Give a presentation in a local meetup or your workplace. Writing is slow but the responses of readers even slower. Test your ideas in the faster, less permanent medium of speech, like an presentation, before you try to set them down in the medium of writing.

This is a nice checkpoint. If you're not confident if people are willing to listen to what you're saying, what makes you think people are willing to read a book about this topic? Each slide heading is a subtopic, simply a thought from you, the writer. Any word on the slides are the key points of your thoughts. When it comes to step #2, you can just copy-paste the slide and clarify the key points, improve what you are intended do say, to write.

Step #2: Typing

This is what people often refer to writing and are scared of. But what it's definively not: writing. It's just typing. In compare to the final product mindless, easy typing. Anyone can do it. Modern people using email quite often, many all the day. People does not expect great literature, just info.

You paste the TOC (or the slide content) and fill in the blanks. Add more explanation about what the meaning is, what do like to express with this item in your list.

Douglas Crockford, window="_blank" said in a talk that if you take all the code you've written in a year and just retype it, it will take only a day or two. Three. A week. Long, unpleasant days. But just days. So what are you doing the rest of the year? Definitely not typing.

The typing is to writing what a hackathon is to software engineering. The outcome is a hack but quick, fast. It's ugly but it works. It exists where it didn't exist yesterday. You've created something. Then you have to go back and make it right, rock solid and properly engineered.

Coding and writing technical books have similarities, but there's one difference: you can run the code and see if it does what it's supposed to do. With a book there are many right answers and outcomes that work. You just have to do enough of it to discover your opinion and feel for what works. The book is like source code that doesn't print anything out. So you're working to make the source code of the book readable for others and maintainable for yourselves.

It's very important that you don't read or edit at this stage. No formatting. No editing. Just typing away, just filling the blanks. Like filling a form. That's why I called it mindless.

> If you type on pages, the headline is the important part. If you haven't additional words on the topic present, use blind text!Ñ!so-called lorem. Your AsciiDoc toolset supports blind text that is automatically generated. No need to fear for the White Page.
>
> *A lorem block look something like this*
>
> Voluptatem temporibus atque dicta et dolorem qui nostrum sit quam in tenetur enim. Itaque temporibus porro totam rem ut eos ut officia odit praesentium perspiciatis. Reiciendis vero quia corporis et dicta quo quidem minima cum omnis. Quisquam quia porro libero tempora id consectetur.

Back to writing or reading email. Most people are OK with writing emails. They don't consider on this writing, reader's stand for mistakes or missing words. There's no writer's block for email - not needed. Someone asks you a question, you reply and type away. An email has a subject. What you're writing about has a subject (like the TOC items). It's just like answering a question.

Next TOC item.

And the next one.

Before you know it, you've typed in lot. Resist the temptation to do editing. Keep typing.

I found it really helpful to spend some time in the morning and type for half an hour. It's awesome. It's the morning and you already have something done. Something is accomplished. Words are down on paper (of course you haven't read the words and you have no idea how bad the writing is, and this is a good thing).

You feel good the rest of the day. Even if the rest of the day is just busy work or sitting in meetings or stuck in the traffic: You have your morning, you have been productive already. In the night you may write more or you may not. You may enjoy going out or just being lazy.

So think of this stage as just typing which is easy and brainless. But productive and save your mood. Have fun, don't underestimate this. A brain is tricky part of your body. Never punish. No bad vibrations on what you would like to do.

Step #3: Editing

This is a read-write loop: you read what you wrote and you fix it. You move stuff around. You add, you remove, you refactor. Then you do it again.

You may be surprised initially that the stuff what you wrote in step #2 looks like, let me quote Frank Zappa here, *some of the most terriblest shit you've ever known*. But it's fixable. It's down on paper. Writing's done, you have pages and pages of text. You're an author! The rest is editing. And you can do it anywhere.

The typing is the only part that requires you to sit down in front of a screen. Thinking (the TOC) is done outside the four walls. I do most of my editing outside too. By the pool, the beach, if I'm walking, even for the morning procedure. I can do it in while in line at the store. You just print what you wrote and scribble on top of it.

I suggest the first edit is on the computer though because there will be too much scratching.

You can do the formatting in this stage. And print and edit. For some reason it's easier to spot mistakes in printed words. If you're into grammar checks you can also read backwards. For me, English is not my mother tounge, so I don't even try to fix grammar. That's you have an editor. If you don't have a publisher, I hear you can hire an editor.

Then there's the technical review, which is also critical. You try to find the best people on the topic to agree to read your stuff and point your mistakes. Because mistakes you will make. Fact of life, no matter how smart you are. So then another round of editing. Then the grammar Nazi kicks in and fixes your commas. And another round of editing.

The blog hack

Blogging is a nice way to hack writing; to do writing quite fast. You don't need a whole TOC, just one item. Then you type. Then you hit "publish". (ok, maybe read it once first, but then publish immediately). Then you edit if you want. No need to do much editing though. You have a feedback loop from the readers, if people point out mistakes in your logic, code, writing or whatever, it doesn't matter how polished and grammatically correct it is. You can edit the technical part first which is the most important. A technical book should first and foremost be technically correct.

Repeat the blog hack enough times on related subjects and voila, you have a series of posts and in the very end: a book. You only need to tighten it up and do some more of editing.

All in all you'll spend more time editing than typing. Editing takes time. That's why you keep it off the typing phase. It takes time but it's not writing. Again, no writer's block here. You can do it outside in the Nature.

Whatõs next

You read, more or less, a lot of theory. Thatõs fine, good to know!Ñ!needed to know. Thatõs why itõs written. Simple like this.

Now, it time to make things real. No longer only reading. Now go back to work! But: Stop writing and start typing already!

Editor - The Swiss Army Knife

Suscipit voluptatem repudiandae consequatur beatae eius eius et explicabo totam enim error libero. Quo sint vel qui ut dolorem consequatur laboriosam aliquid explicabo est consectetur unde fugit. Est ipsa voluptatum beatae. Et earum consequatur sint est sapiente fugit. Dignissimos provident odit laudantium sed et omnis fuga et nostrum ut est blanditiis.

Figure 2. Editor Atom

Possimus ut et saepe. Dicta voluptatum ducimus et nemo aut qui eum aliquid aliquam numquam.

Building documents

Coming back to chapter Non disturbing Techniques, the Act of Writing includes a lot of very formal, more ore less mind-less activities. Sounds quite boring but it̊s not. Many people have fun things to do like ironing. Yoůll have nice results, and your can wander your thoughts, even watching TV. That is really great, the beginning of what I call the Flow.

The Flow is important. As much as possible prevent situations you get stuck. For sure, you will experience getting stick on searching a good explanation, good words - searching for a good idea. You will need what̊s called inspiration!Ñ!but cannot force them. For that reason calm you down: a lot of formal activities are waiting that needs to be done.

Enough space for the Flow.

The name of this chapter is Building documents!Ñ!that̊s not for chance. The name Building documents should imply what is means: build a document, no writing. No brick layer goes for thinking what a brick is, how create them. There are already bricks available. Otherwise it wouldn̊t possible to build that house.

Think along that picture: building a house is the base, but to get a house furnished a completely different thing. Who has ever build a house will remenber very different phases of that project. If the construction is finished, a another quite exhausting job is waiting for you. Make that house a living space for you and your family. That situation has quite a lot of similarities to what I called editing.

Like the construction of a house, building documents needs many different things to do. Different craftspeople were morking on that house; from the roof to the cellar. Quite the same to building documents. But: you are the builder, no experinced craftspeople will take care.

What̊s needed to build the base of a document, no visionary is needed. Houses are different but using the same components. For that reason I created a raw base for that: the document skeleton.

A starter site created by the *J1 Template* (gem) includes the skeleton. Located in root folder of such a site. Yoůll find the skeleton as zip file: asciidoc_document_skeleton.zip under the directoty _templates.

Now, it̊s time to make your hands dirty! Go for The Document Skeleton.

The Document Skeleton

The Asciidoc Document Skeleton a helper for setting up a base file and folder structure for multi-chapter AsciiDoc documents based on te Web site generator *Jekyll* and *J1 Template*, the base system creating a J1 Web site.

!

Converting the Skeleton

The Asciidoc Document Skeleton is fully relocateable and can be placed in any subfolder of your Jekyll site for HTML output. For PDF output. A single variable `BASE_PATH` has to be set for your environment.

See the batch helper files `a2p.bat` for *Windows* or `a2p.sh` for the shell script used on *Unix* or *Linux* OS (bash).

Figure 3. Asciidoc Document Skeleton

Install the Skeleton

From now on, you need to have the base software components *Jekyll* and *J1 Template* in place for developing. If you haven't installed the softeare already , see the Tip Box below.

#

Explain from where the tools can be downloaded.

To not intefere already existing projects, it might be an good idea to create a new Web for the very first work on that. Nothing simpler like that.

- Open a command box and go for your WWW root folder (or any other directory you like)
- Create your site, change to the site's folders
- Run jekyll

```
cd $WWW_ROOT
j1 generate doc_site
cd doc_site && j1serve
```

You'll see something like this (I did it on *Windows*):

```
C:\Temp>j1 generate doc_site
Running bundle install in C:/Temp/doc_site...
Ê          Bundler: Fetching gem metadata from https://rubygems.org/..........
Ê          Bundler: Fetching gem metadata from https://rubygems.org/..
Ê          Bundler: Resolving dependencies....
Ê          Bundler: Using rake 12.3.0
Ê          ...
Ê          Bundler: Bundle complete! 26 Gemfile dependencies, 89 gems now
installed.
Ê          Bundler: Use `bundle info [gemname]` to see where a bundled gem is
installed.
Generate jekyll site installed in C:/Temp/doc_site.

Configuration file: C:/Temp/doc_site/_config.yml
Ê           Source: .
Ê      Destination: _site
ÊIncremental build: enabled
Ê     Generating...
Ê        AutoPages: Disabled/Not configured in site.config.
Ê       Pagination: Complete, processed 1 pagination page(s)
Ê                   done in 24.679 seconds.
ÊAuto-regeneration: enabled for '.'
Ê   Server address: http://localhost:40000
Ê Server running... press ctrl-c to stop.
```

After 30 seconds I'm done. Open a browser pointing to the new site doc_site displayed in site-creation output (look for `Server address` at line 22).

Figure 4. J1 Starter Web

Run your favourite editor and open the Web site folder. Im my case, I created the site under C:/Temp/doc_site/. I personally using for *Atom* to edit my Jekyll sites. A quite good one, created by *Google* and and many, many people from the community adding a huge number of addon̈s (so-called Packages). To read and check my AsciiDoc code better, I installed a syntax highlightning package.

For that reason, if a screenshot is placed to make things more visual, yoüll see the Atom editor. But as I said, use what you want, have ever used or go for an editor you prefer. Using a tool you like supports the Flow.

Extract the skeleton from the zip file to the pages folder of doc_site. Thaẗs the home for classic web pages for a Jekyll site. The first file to inspect should be a so-called Master Document the AsciiDoc file document_title.adoc.

Figure 5. Master Document for HTML conversion

You did a lot of things already over the time working on that tutorial. I’ve no idea how long. That is the author’s destiny.

A suscipit est rerum. Eveniet illum unde magni adipisci provident reiciendis sint repellendus similique consequuntur.

Files and folders of the Skeleton

- Explain the general folder structure
  - include folder
  - include documents
  - AsciiDoc attributes
  - numbering scheme
- Introduce document types
  - entry documents
  - chapter documents -

AsciiDoc Preprocessor

The AsciiDoc Preprocessor used by Asciidoctor is a powerful component of the AsciiDoc language engine to organize source files for AsciiDoc documents. The main pupose of the preprocessor is:

- partitioning large documents into smaller files
- load parts|snippets of a source file if used multiple times (e.g. examples)
- create document variants for different use (e.g HTML or PDF output)

Including Files

The Preprocessor is a file merger. Based on the `include::` directive, the preprocessor performs a simple file merge before the lines are parsed by the language engine into a document structure.

The `include::` directive looks like a AsciiDoc block macro, but it is a preprocessor directive like `ifdef` or `ifeval`. That means no environmental information (attributes) are evaluated. To control a file merge, see chapter Conditional Processing.

Merging files works with any sort of text files, not only files containing Ascidoc data. The content of all included files is normalized. What means, the encoding of the data is forced to UTF-8 or converted from UTF-16 to UTF-8 if the file contains a BOM. Trailing whitespace and endlines are removed from each line and replaced with a Unix line feed.

The files recognized as AsciiDoc files has one of the following extensions:

```
.asciidoc .adoc .ad .asc .txt
```

Asciidoctor runs the preprocessor on all the lines the `include::` directive is placed, loads the data and interprete the following directives:

- includes
- preprocessor conditionals (e.g., `ifdef`)
- front matter (if enabled)

This allows including files to be nested, and provides lot of flexibility in constructing quite different documents with a single master document and a few command line attributes.

Conditional Processing

You can include or exclude lines of text in your document using one of the following conditional preprocessor directives:

- ifdef
- ifndef
- ifeval

These directives tell the processor whether to include the enclosed content based on certain conditions. The conditions are based on the presence or value of document attributes.

For example, say you want to include a certain section of content only when converting to HTML. Conditional preprocessor directives make this possible. You simply check for the presence of the `backend-html5` attribute using an `ifdef` directive.

Checking multiple attributes

Both the `ifdef` and `ifndef` directives accept multiple attribute names. The combinator can be 'and' or 'or'.

Any attribute (or)

Multiple comma-separated (,) directive names evaluate to true, allowing the content to be included, if one or more of the directives is defined. Otherwise the content is not included.

Any attribute example

```
ifdef::backend-html5,backend-docbook5[Only shown when converting to HTML5
or DocBook 5.]
```

All attributes (and)

Multiple plus-separated (+) directive names evaluate to true, allowing the content to be included, if all of the directives are defined. Otherwise the content is not included.

All attributes example

```
ifdef::env-github+backend-html5[Only shown when converting to HTML5 on
GitHub.]
```

The `ifndef` directive negates the results of the expression.

Starting in Asciidoctor 1.5.6, the operator logic in the `ifndef` directive changed to align with the behavior of AsciiDoc Python. Specifically, when attributes are separated by commas, content is only included if none of the attributes are defined. When attributes are separated by multipe plus characters (+), content is included if at least one of the attributes is undefined.

See issue #1983 to find the discussion about this behavior and the rationale for the change.

`include` Directive

The include directive has the following anatomy:

```
include::path[leveloffset=offset,lines=ranges,tag(s)=name(s),indent=depth]
```

The leveloffset, lines, tag(s) and indent attributes are optional, making the simplest case look like:

```
= Document Title

include::content.asciidoc[]
```

The sections that follow go into detail about how the include file is resolved and how each attribute is used.

File resolution

The path used in an include directive may be either relative or absolute. If the path relative, the processor resolves the path using the following rules:

- If the include directive is used in the main (top-level) document, relative paths are resolved relative to the base directory. (The base directory defaults to the directory of the main document and can be overridden from

the CLI or API).

- If the include directive is used in a file that has itself been included, the path is resolved relative to the including (i.e., current) file.

These defaults makes it easy to reason about how the path to the include file is resolved.

If the preprocessor cannot locate the file (perhaps because you mistyped the path), you'll still be able to convert the document. However, you will get the following warning message during conversion:

```
asciidoctor: WARNING: master.adoc: line 3: include file not found:
/.../content.adoc
```

The following message will also be inserted into the output:

```
Unresolved directive in master.adoc - include::content.asciidoc[]
```

To fix the problem, edit the file path and run the converter again.

If you store your AsciiDoc files in nested folders at different levels, relative file paths can quickly become awkward and inflexible. A common pattern to help here is to define the paths in attributes defined in the header, then prefix all include paths with a reference to one of these attributes:

```
:includedir: _includes
:sourcedir: ../src/main/java

include::{includedir}/fragment1.asciidoc[]

[source,java]
----
include::{sourcedir}/org/asciidoctor/Asciidoctor.java[]
----
```

Keep in mind that no matter how Asciidoctor resolves the path to the file, access to that file is limited by the safe mode setting under which Asciidoctor is run. If a path violates the security restrictions, it may be truncated.

Partitioning documents

When your document gets large, you can split it up into subsections for easier editing as follows:

```
= My book

include::chapter01.asciidoc[]

include::chapter02.asciidoc[]

include::chapter03.asciidoc[]
```

| Note the blank lines before and after the include directives.

This practice is recommended whenever including AsciiDoc content to avoid unexpected results (e.g., a section title getting interpreted as a line at the end of a previous paragraph).

The leveloffset attribute can help here by pushing all headings in the included document down by the specified number of levels. This allows you to publish each chapter as a standalone document (complete with a document title), but still be able to include the chapters into a master document (which has its own document title).

You can easily assemble your book so that the chapter document titles become level 1 headings using:

```
= My Book

include::chapter01.asciidoc[leveloffset=+1]

include::chapter02.asciidoc[leveloffset=+1]

include::chapter03.asciidoc[leveloffset=+1]
```

Because the leveloffset is *relative* (it begins with + or -), this works even if the included document has its own includes and leveloffsets. If you have lots of chapters to include and want them all to have the same offset, you can save some typing by setting leveloffset around the includes:

```
= My book

:leveloffset: +1

include::chapter01.asciidoc[]

include::chapter02.asciidoc[]

include::chapter03.asciidoc[]

:leveloffset: -1
```

The final line returns the leveloffset to 0.

Alternatively, you could use absolute levels:

```
:leveloffset: 1

//includes

:leveloffset: 0
```

Relative levels are preferred. Absolute levels become awkward when you have nested includes since they aren't context aware.

`ifdef` Directive

Content between the `ifdef` and `endif` directives gets included if the specified attribute is set:

ifdef example

```
ifdef::env-github[]
This content is for GitHub only.
endif::[]
```

The syntax of the start directive is `ifdef::<attribute>[]`, where `<attribute>` is the name of an attribute.

Keep in mind that the content is not limited to a single line. You can have any amount of content between the `ifdef` and `endif` directives.

If you have a large amount of content inside the `ifdef` directive, you may find it more readable to use the long-form version of the directive, in which the attribute (aka condition) is referenced again in the `endif` directive.

ifdef long-form example

```
ifdef::env-github[]
This content is for GitHub only.

So much content in this section, I'd get confused reading the source without
the closing `ifdef` directive.

It isn't necessary for short blocks, but if you are conditionally
including a section it may be something worth considering.

Other readers reviewing your docs source code may go cross-eyed when
reading your source docs if you don't.
endif::env-github[]
```

If you're only dealing with a single line of text, you can put the content directly inside the square brackets and drop the `endif` directive.

ifdef single line example

```
ifdef::revnumber[This document has a version number of {revnumber}.]
```

The single-line block above is equivalent to this formal `ifdef` directive:

```
ifdef::revnumber[]
This document has a version number of {revnumber}.
endif::[]
```

ifndef Directive

`ifndef` is the logical opposite of `ifdef`. Content between `ifndef` and `endif` gets included only if the specified attribute is *not* set:

ifndef Example

```
ifndef::env-github[]
This content is not shown on GitHub.
endif::[]
```

The syntax of the start directive is `ifndef::<attribute>[]`, where `<attribute>` is the name of an attribute. The `ifndef` directive supports the same single-line and long-form variants as `ifdef`.

`ifeval` Directive

Content between `ifeval` and `endif` gets included if the expression inside the square brackets evaluates to true.

ifeval example

```
ifeval::[{sectnumlevels} == 3]
If the `sectnumlevels` attribute has the value 3, this sentence is included.
endif::[]
```

The `ifeval` directive does not have a single-line or long-form variant like `ifdef` and `ifndef`. The expression consists of a left-hand value and a right-hand value with an operator in between.

ifeval expression examples

```
ifeval::[2 > 1]
...
endif::[]

ifeval::["{backend}" == "html5"]
...
endif::[]

ifeval::[{sectnumlevels} == 3]
...
endif::[]

// the value of outfilesuffix includes a leading period (e.g., .html)
ifeval::["{docname}{outfilesuffix}" == "master.html"]
...
endif::[]
```

Values

Each expression value can reference the name of zero or more AsciiDoc attribute using the attribute reference syntax (for example, `{backend}`). Attribute references are resolved (substituted) first. Once attributes references have been resolved, each value is coerced to a recognized type.

When the expected value is a string (i.e., a string of characters), we recommend that you enclose the expression in quotes.

The following values types are recognized:

number
: Either an integer or floating-point value.

quoted string
: Enclosed in either single (') or double (") quotes.

boolean
: Literal value of `true` or `false`.

How value type coercion works

If a value is enclosed in quotes, the characters between the quotes are preserved and coerced to a string.

If a value is not enclosed in quotes, it is subject to the following type coercion rules:

- an empty value becomes nil (aka null)
- a value of `true` or `false` becomes a boolean value
- a value of only repeating whitespace becomes a single whitespace string
- a value containing a period (`.`) becomes a floating-point number
- any other value is coerced to an integer value

Operators

The value on each side is compared using the operator to derive an outcome.

`==`
: Checks if the two values are equal.

`!=`
: Checks if the two values are not equal.

`<`
: Checks whether the left-hand side is less than the right-hand side.

`<=`
: Checks whether the left-hand side is less than or equal to the right-hand side.

`>`
: Checks whether the left-hand side is greater than the right-hand side.

`>=`
: Checks whether the left-hand side is greater than or equal to the right-hand side.

The operators follow the same rules as operators in Ruby.

Index