Guide to Editing at DevCAD

This guide attempts to give the members of the Developer Content and Advocacy Team (DevCAD) an orientation to the role of the editor and a guide to the editorial process. As opposed to a style guide or grammar primer, this guide is about what it means to be an editor (*or play one on TV!*), how the editor brings value to the publishing process, and how to go about executing the editorial role. In that respect, it attempts to distill my nearly 40 years of experience and understanding of my role into a brief orientation that anyone with a reasonably good grasp of writing principles, grammar, and punctuation could use to do my job. Wish me luck!

Contents

Executive Summary (TL;DR)

- **Editing isn't writing; it's making writing better** (BASF commercials!). Editorial work corresponds to code refactoring/optimization, rearchitecting, and testing in the software development world. The most critical skill of the editor is to approach the work from the perspective of the reader: not to judge the copy based on your own knowledge of the material or familiarity with the writer, but to empathize with readers and anticipate what they do and don't bring to their own reading of the content.
- **No one can be their own editor.** First, you can't judge the clarity and logic of your work because, since you wrote it, of course it seems good to you. Only another person can approach your work from the perspective of the reader and give you critical feedback. Second, because you wrote it, you know what's supposed to be there and you won't catch the typos; only someone else, who hasn't got that preperception, can see what's actually on the page.
- **No, AI can't do the job.** Don't get me started. Grammar checkers can't tell whether the author has covered the topic adequately, taken into account their readers' prior knowledge or lack of same, or focused the presentation on their readers' needs and interests; some of their recommendations are also simply wrong. AI analysis can't do this, either.
- **The goal of editing, at least in the context of technical documentation, is clarity.** Above all, writing must communicate clearly. This means that any written work must have well-

defined goals, a logical order to its points, and unambiguous statements of the ideas it's trying to express.

- **Editing well requires consistency.** A lot of editorial choices are a matter of personal taste as much as proper grammar, so the key skill of editing is to be able to remember the editorial decisions you make as you go through a piece, and on to the next piece, and to make those same editorial decisions *the same way* every time you encounter the same editorial issue. Style guides are the codification of these accumulated individual editorial decisions; use style guides rigorously to ensure editorial consistency across our entire published work.
- **The standards codified in style guides are subject to the context of the work:** who is its author, who is its audience, what are its goals, what is its format and structure, and where will it be published? These considerations all have an impact on the editorial decisions you make.
- **Prerequisites for successful editing** are a thorough understanding of the standards of English grammar, punctuation, and typography, specific to the context of the work: some usages are perfectly fine in a blog post, but not in a Developer Center page, for instance. Another underappreciated prerequisite is an understanding of web rendering technologies (HTML, Markdown, etc.) and how they will affect the way the content is rendered.
- **Before you can edit any piece of content, you need to understand its complete context:** the target audience, the goals of the writer, the content model (if any), the subject matter itself, and the proposed publishing location.
- **Do no harm.** In any context, make the minimal changes to the copy required to serve the needs of clarity, concision, structure, and grammar. Make sure your edits introduce no changes in meaning, create no ambiguity, add no unnecessary verbiage, nor change the author's voice (where voice is appropriate).
- **Voice and tone are different.** Be aware of whether or not it's appropriate for the piece in question to have an individual voice: technical documentation does not, but blog posts, community posts, and any other pieces with a credited author do. Voice is different from tone, however, and edits to enforce tone in the context of technical documentation may be neither necessary nor appropriate when editing a piece with a voice.
- **Familiarize yourself with the most common errors** of grammar and punctuation in modern published writing (trust me, there are fresh examples every day) and be on the lookout for them in every piece you edit.
- **Develop a consistent approach to the process of editing.** Some editors do everything in one pass while others make multiple passes to cover different types of issues they typically encounter. You may decide to edit a portion of a document in one sitting and come back later to do the rest; or you may stay focused on the work to the exclusion of all other tasks until you've completed it. If the former, develop habits of tracking your progress so you don't miss parts you haven't edited.
- **Software tools for editing provide power, not steering, and can easily be misused.** WARNING: There is no context in editing in which it's safe to globally search and replace

any string with any other string without the danger of introducing incorrect or inappropriate edits. If you use S&R, step through the piece and examine each proposed replacement as you go. Trust me, you'll be surprised. Be aware also that the spelling checker and Grammarly recommendations aren't flawless either; they can and do sometimes recommend incorrect "corrections". Never accept what they offer without reading the surrounding context first. I've yet to identify an editorial task for which AI tools offer any practical value; if you try AI, be skeptical and examine its work minutely.

- The editorial process at DevCAD looks like this:
    - It takes place AFTER technical review issues have been resolved and the draft is, as far as the author is concerned, final.
    - Work in Google Docs (or sometimes, in the case of API Reference pages, in PR review) and track your edits as suggestions to the copy.
    - It's the author's responsibility to resolve your suggested edits. As a matter of course, they should accept these, but as the author (and presumably more of an expert than you are on the subject matter) they reserve the right to reject edits they feel are inappropriate, or that introduce error or ambiguity.
    - Once the author has resolved the edits, they are free to proceed with the rest of the publishing process. You may be asked to review the produced work in site preview before publication, but that should be merely to verify the quality of the production, not to re-edit the content.

Why editing?

Why do writers need editors? As Robert Graves once said, "There is no such thing as good writing, only good rewriting." Put another way, editing isn't writing; it makes writing *better*. (Quick trip down memory lane: BASF commercials!). Editing is more than "fixing" grammar, spelling, and punctuation errors; it's the comprehensive consideration of the quality of the written work in the context of its subject matter, goals, intended audience, structure, and publishing context, and optimizing its quality to serve those purposes best.

So, what does quality consist of with respect to technical writing? First, it does include proper grammar, spelling, and punctuation, of course. These days we all make fewer mistakes in these areas, because that's where spelling and grammar checkers do work (*mostly*) well. But beyond those basics, quality in writing consists of clear meaning, logical flow, concision, consistent tone and style, and impactful delivery that commands the reader's attention and firmly implants its ideas in the reader's mind.

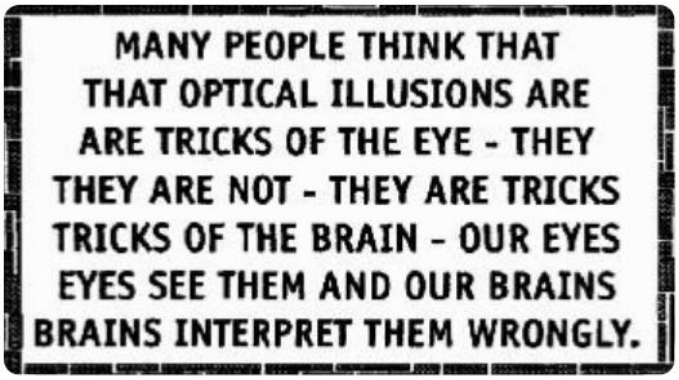
Core editorial principles

Here are the principles on which editorial practice is founded:

Everyone needs an editor

No one can edit their own work. There are two main reasons:

- Because you wrote it, you are predisposed to view it as good. After all, when you created the work, you put in every effort to make sure it was technically accurate, explained your points clearly and in good order, anticipated the prior knowledge of the reader and explained what you anticipated they wouldn't know, and so forth. The problem is, when you write, you're essentially talking to yourself; and since you already understand yourself, you understand not only what's in the text, but what you meant by it: what's not said, what's implied, and what you're referring to in every context. There's absolutely no guarantee any of this is true for your reader; in fact, it's almost guaranteed there will be at least some point you're making, or something about the way you say it, that will be ambiguous or obscure in at least some readers' minds. Only another person reviewing your work can be aware of these cases, because they don't carry your prior knowledge into the reading of your work.
- It's really hard to spot all your typos and other mistakes when you're looking at your own work. It's like this old optical illusion:



  You've probably spotted the issues here, but the point the text is making is also valid, especially when you're looking at your own work. Because you wrote it, you know what you were saying, so you know what's supposed to be there and your brain fills in for those spots where it's not. Only another person, who brings fresh eyes and brain to your work, can spot your mistakes reliably.

Clarity is the goal

The goal of editing, at least in the context of technical documentation, is *clarity*. The purpose of technical communication is to inform and instruct: to present concepts and procedures in such a way that the reader will understand them clearly and completely. Every decision an editor makes should be justifiable with that goal in mind: it should make the writing clearer, more accurate, and easier for the target audience to understand.

Consistency is the method

Editorial consistency is simply the practice of making the same editorial decisions the same way every time you encounter the same issues while editing, not only in the current piece of writing, but in every piece of writing that you edit.

This isn't as straightforward as it may seem, because the rules of proper English grammar, punctuation, spelling, and usage allow for a lot of possible solutions to the same editorial problem; a given editorial issue may be fixable in any number of ways that perfectly pass those rules. Here's the dark secret of editing: a lot of editorial choices are a matter of personal taste as much as proper grammar. While an editorial choice you make may be perfectly defensible, so may another choice on the same text, and which way you choose to make that edit reflects your predispositions as much as it does the concern for correct and clear expression. So, the key skill of editing is to be able to remember the editorial decisions you make as you go through a piece, and on to the next piece, and to make those same editorial decisions the same way every time you encounter the same editorial issue.

When working on any significant work, editors often keep a style sheet: a simple list of notes recording particular editorial decisions they've made, so they can refer to it and make the same edit when they see the situation crop up many pages later. They keep those style sheets and apply them to future documents as well. Out of this practice arises a consistent approach to editing that produces a characteristic *editorial style* to the publisher's work.

From consistency comes editorial style

Editorial style, as it arises from editorial practice, is central to ensuring quality across a publisher's content set. As any editor's approach to a given piece of writing will be as individual as they are, it's important that every editor on a team, or in an organization, follows the same editorial style guidelines. These guidelines are codified in a master document: house style.

Context determines standards

House style guides don't cover everything, the way a more general reference work on editorial standards tries to. (No one work can cover everything; that's why every professional editor uses multiple editorial references.) But the context of the writing you're working on determines the standards used and how they're applied. We do technical documentation, so we've chosen standards that apply in that context. Our standards would be very different if the context were poetry, journalism, or fiction.

One specific context closely related to ours that deserves mention is that of marketing communication. We work closely with Developer Marketing and are sometimes tasked with producing content that will serve marketing purposes, either instead of or in addition to those of

technical documentation. It's important to remember which context you're working in, either as a writer or editor, and make sure your work optimizes the content for the purpose(s) it serves.

Editing in practice

To work as an editor, you need some fundamental aptitudes, a lot of domain knowledge about language, and skills borne of practice. The following is merely a distillation:

Prerequisites

It should go without saying that you can't be a good editor if you're not first a good writer. However, being a good writer doesn't make you a good editor as well. Editing is a distinct discipline from writing, with its own qualifications, and foundational to being an effective editor are the following strengths:

An eye for detail that sees exactly what's in the document without letting the brain gloss over or fill in what's not there, or what's there that's wrong.

An understanding of what's clear and the ability to spot ambiguity, especially when you're most familiar with the topic of the piece. This is the essential quality you need to be able to place yourself in the position of the reader, so you can understand how to clarify passages that may seem perfectly clear already, both to you and to the author.

An ear for eloquence. Just because a particular passage is grammatically correct, or even reasonably clear, that doesn't necessarily make it good writing. Good writing possesses energy and impact. Consider this example from Strunk and White (one of my favorites). One of the following sentences has gone down in American history and its author is forever remembered; the other would have been forgotten forever instead:

These are the times that try men's souls.

Soulwise, these are trying times.

For another example of the power of eloquence (and how to kill it), try the infamous Gettysburg PowerPoint Presentation.

While these qualities of the intellect are foundational to editing, you also need specific knowledge and skills to do the work: hence the following sections.

Composition and organization

Here, I'm lumping all the principles of writing that operate at a larger scale than the sentence. Things to know:

Outlining: How to organize material into topics and subtopics and present them in logical order while maintaining consistency, unity, coherence, and emphasis.

Sectioning: How to group material into sections that group a coherent set of related points

Paragraphs: How to organize sentences into paragraphs: what a topic sentence is and where to put it, how to judge whether other sentences in the paragraph support the topic, and how to keep irrelevant statements out of the paragraph entirely.

Grammar

Of course, it goes without saying that editors enforce proper grammar. But to what standard? Every publisher makes this choice about everything they publish. At DevCAD, we follow the same standards of most modern technical publishers: formal standard written English (FSWE), with informal variations. We allow the use of:

- a conversational tone
- *you* to address the reader
- *they* and its related pronouns to refer to singular subjects
- common (but not colloquial or dialect) contractions (no *ain't*, for instance)
- ending a phrase or sentence with a preposition when it would be awkward to construct the sentence formally (This is definitely a judgment call.)
- in most cases, split infinitives

However, we don't allow the following:

- Slang
- Colloquialisms
- Dialect
- Nonstandard English
- The use of *we* in reference to the author and the reader

Finally, there are a few artifacts of formal writing we discourage:

- Latin-based abbreviations (*i.e.*, *e.g.*, *etc.*, and so on)
- Places where enforcing formality induces awkward phrasing (up with which I will not put!)

To edit, you need a thorough understanding of English grammar, including (but not just) these concepts:

- Parts of speech
- Parts of a sentence
- Tenses, cases, and moods
- Sentence structure and its variations

- Subject-verb agreement
- Types of verbs: transitive vs. intransitive, regular vs. irregular
- Parallelism
- Clauses vs. phrases
- Antecedents
- Appositives

You probably haven't had to think explicitly about these concepts since you got out of school, but you need to be aware of them, identify them in the text you edit, and recognize problems with them.

Punctuation

Punctuation rules abound on the internet; there's no need to elaborate much here. A few points:

- Know the proper uses of each punctuation mark.
- Learn when (and when not!) to use a comma.
- Learn the distinct uses of en and em dashes vs. hyphens. At DevCAD, we don't use spaces around dashes, and we don't use the double hyphen; that's only valid if you're on a typewriter (?!) or as Markdown syntax.
- The space-hyphen-space (-) is simply wrong in all contexts; *never* use it.
- Don't use a dash in place of a colon.
- Do use the Oxford comma. This is one instance where we differ from Docusign corporate style. Corporate follows the AP Stylebook, which forbids the Oxford comma; that makes sense in marketing, where you want to be less formal and more energetic, but in technical documentation, leaving it out often induces ambiguity.

Typography

Typography impacts our work to a limited degree; mostly, we avoid typographic issues entirely. Our goal is to create work that's clean from a typographic point of view, so that it imports cleanly into our content management systems and expresses our intent clearly on the final page.

A few points:

- Put only one space after a period (full stop, as the English have it). Two spaces after a period is an old typewriting practice.
- Except for using spaces to indent code, there should never be more than one space separating anything from anything in our content. Do *not,* however, do a global search-and-replace for this, as you will screw up the code indents that way.
- Use bold and italic sparingly, because they have particular uses in the context of our docs; see our Developer Content Style Guide for rules.
- Also see the Style Guide for use of code font.

- Be sure to check for the use of proper quote marks in regular text (“curly” quotes or “smart” quotes). Google Docs lets you choose to insert the correct quote marks when you write or edit (**Tools > Preferences >** “Use smart quotes”). Vertical quote marks are only for code.
- Bulleted or numbered lists: The difference between a bulleted list and a numbered list is the indication of *order:* sequence, importance, or some other modality. This may seem obvious, but you’d be surprised at the number of times this comes up in editing. If the items in the list have no order, they should be bulleted; conversely, don’t use bullets for a list that should be ordered! This consideration applies individually at every level of a hierarchical list structure: at each level, if an order is present, use an ordered list, and if it’s not, use bullets.

Understanding the reader

One of those most essential skills an editor can have is to empathize with readers and anticipate what they do and don’t bring to their own reading of the content. You, as the editor, must adopt the point of view of the reader and judge the quality of the work from the reader’s perspective. After all, the ultimate goal of any documentation is to communicate information to the reader; the document may be fully detailed and technically correct, but if it’s full of terminology the reader isn’t familiar with, assumes concepts the reader doesn’t know, or is simply written in too dense and complicated language, how is the reader going to understand it?

One easy way to recognize potential problems is to be aware of whether you yourself understand the content; you can’t assume a reader will be able to follow along if the author’s presentation confuses you. Given that at DevCAD, we’re all familiar with Docusign technology, however, you also can’t assume the reader will be able to follow along just because you can! Ask: if I didn’t already know what all these things are that the author is referring to, would I understand what they’re saying? What would a reader new to Docusign need to know to understand this, and how can we make sure they can find out?

Note that this is entirely separate from the question of whether the content is grammatically correct or punctuated correctly. A document can be flawless in both those respects, but impossible for all but an expert (or even the author) to read! Reviewing a piece of content and passing it on the basis of its being grammatical is to abdicate your responsibility to the reader to make sure it’s understandable.

So, who is our reader? At Docusign, we assume the reader:

- is fluent in English and has at least an eighth-grade vocabulary;
- is generally educated and can be assumed to know common things you learn in school or from common internet sources;
- has general programming knowledge and is familiar with not only common software development concepts, but also the specifics of developing, testing, and distributing applications in their programming language and environment of choice;
- and most importantly, they may have no prior background in Docusign technology at all.

This has obvious implications for how we write content, and therefore how we edit. When you review a piece of content, always make sure that concepts specific to Docusign are either adequately explained or cross-linked to, or that the concept under consideration is covered on a page within the immediate neighborhood of the location where this content will be hosted: on an adjacent page or the parent page on the page in question.

Understanding the goals of the content

All content has goals: to tell a story, to impart information, to instruct, to persuade, to inspire emotions, and more. You need to understand the general as well as the specific goals of the content.

At DevCAD, we deal with documentation, which always seeks to inform, and sometimes to instruct; we also occasionally deal with marketing content, which seeks to persuade. Beyond these generalities, however, you need to understand the specific goals of the content: *what information* does it seek to deliver? What *skills* does it seek to impart? What *emotions* is it trying to inspire? Ask yourself, before you edit anything: *What constitutes success for this content?* If you can express that in a single statement, you will understand the goals of that content, and you can edit the text to ensure it achieves them.

Understanding the content model

All technical documentation on the Developer Center is written to a content model:

- How-tos follow a fairly templated and prescriptive model, to the point of having a lot of boilerplate text.
- API Reference pages follow a structural model, and while there's much less general use of boilerplate, there's a lot of reuse of standardized descriptions for objects, properties, methods and attributes.
- Guides are the most open-ended, but still contain standardized structural elements.

In all these cases, you must be aware of the model into which the work fits and compare it to other pages that parallel the one you're working on. Ask yourself:

- Does this piece have all the structural elements it should have based on its model?
- Are any elements of this piece not according to the model? Is there a good justification for that, or should they be modified to fit the model? This is a good point on which to query the author for their intent and any technical context you need to understand their choices. There may be very good reasons for those choices that you need to know before considering changing the structure of the work.
- For those elements of the model that use boilerplate text, is that text used correctly here?
- Are any elements of this work's content suitable for being replaced by boilerplate?

Understanding voice: creating vs. preserving

All writing has a voice. Voice is that quality of writing that lets you imagine the personality of the author: their tone, their perspective, their choice of words, their characteristic phrasing and diction, and more. It comes out best in an author's work when they express themselves in a way that comes naturally to them, without making conscious choices of style.

Here's the thing: In technical documentation, the author's voice is *impersonal*. This may sound contradictory to the previous paragraph, but it's true: when done properly, documentation doesn't give the reader an impression of an individual author. It's not supposed to; the author isn't important in this context. If our documentation can be said to have an author, it's Docusign itself. So, when you're editing technical documentation for the Developer Center, whether it's a guide, how-to, or API Reference page, it's perfectly fine to edit for that impersonal voice; it's actually desirable. One consequence of this principle is that the word *I* never appears in technical documentation: there is no *I* in the authorial sense.

On the other hand, any context in which an author's name is attached to the work *does* have an individual voice, and that individual voice should be preserved. At DevCAD, this includes blog posts and community posts; there may be other contexts in which an author is credited as well. For such situations, be aware of the author's voice and do not alter it! Thus the next point: *do no harm*.

Do no harm

When editing, it's easy to make the mistake of going too far and, rather than merely making enough changes to optimize the quality of the content, actively rewriting the author's text as you would have written it yourself. "Do no harm", in this context, means that, no matter what you need to do to make a particular piece of writing work for the purpose intended, you need to bring to the work a fundamental respect for the author's intent, and where possible, either preserve their approach to writing it, or rewrite it in a way that anticipates what the author would have done. In the case of technical documentation, this is less about preserving an individual voice (see above) than preserving the anonymous voice of documentation; but in any case, *you must not inject your own voice into the content*. Remember: *If you do your job well, the finished work will seem as if you weren't involved at all*.

There are cases in which extensive rework of an author's prose is required; it often happens when you're working with an author whose first language isn't English, for example. It can sometimes happen when you feel the author has missed major points they should have made, left their points in disorder, or have been frankly incorrect in their description. It most often happens, particularly here in blog posts, when your writer simply isn't very good. However, even in these cases, start by preserving as much of the author's prose as you can, even if you need to rearrange it extensively: keep their diction, look for their characteristic phrasing and reproduce it where possible, and generally try to make your rework sound as much as possible like what the author intended to say.

There's another sense of "do no harm" that's more critical for technical documentation:

Be careful never to introduce inaccurate or incorrect statements
to the work you're editing.

Part of having respect for the author's intent is to have respect for the meaning of what they've written, and never to change that meaning when you're editing. Make sure that your edited version of their prose means *exactly* the same thing as what they wrote: your job is to make it better, not to make it different.

This means two things in practice:

- If you have any doubt that your edits mean the same as what the author originally wrote, add a comment to your edit, assigned to the author, to query them on the meaning and confirm that your edit retains the original meaning.
- If you believe you've spotted something inaccurate or incorrect in the author's work, query them on that point. If you feel confident you can edit their work to correct the issue, do so and invite their review with an assigned comment. *Never* simply edit the work to what you think it's supposed to say without alerting the author to what you've done. After all, you may be wrong!

House style

House style is an authoritative resource maintained in house that codifies all the editorial decisions we must follow specific to our content. House style works at two points in the editorial process:

1. Before you do any editorial work at all, or before you edit a particular piece of content, review the house style in detail so that you have its guidelines in your head as you review the content. In this way, you'll be alert for the issues that the style guide covers and be able to edit in accordance with house style without having to constantly switch back and forth between the work and the style guide, thereby slowing down your process.
2. Refer to the house style guide first for any issues that crop up that you're not sure how to edit.

In addition to referring to the house style guide for any issues you're unsure of, you should also make additions to the house style guide for any issues specific to our content that you don't feel are covered already. The Developer Content Style Guide is a wiki for a reason: all editors are empowered to question its rules or add to them. Add comments to anything you disagree with or are unsure about; the team will be notified of your comment, and (we hope) a useful discussion will ensue.

Using external style standards and resources

House style generally covers just the specific issues that come up often enough in our content to warrant mention; it isn't comprehensive at all. Therefore, house style guides generally refer to broader references that cover the most common general issues as well as obscure edge cases. If you check the Developer Content Style Guide, it references other guides for API Reference material, Docusign Corporate communications, and technical content in general:

- API Documentation Style Guide
- Docusign Copy Style Guide
- Microsoft Writing Style Guide

These references will cover most questions of usage, including terminology. However, more general editorial, grammar, and punctuation questions may require you to refer to other sources. Here are a few I recommend:

- Merriam-Webster Online: The best online dictionary and thesaurus, with good articles on etymology, usage, grammar, and punctuation; a real word nerd joy. If you're wondering whether something should be hyphenated, look it up in the dictionary first: if it's spelled closed in the dictionary, don't hyphenate!
- Grammarly Blog: Lots of great tutorial pieces about usage, style, grammar and punctuation, and good writing.
- Chicago Manual of Style Online: Produced by the University of Chicago Press, this is the online version of their long-standing bible of publishing standards. For most trade and textbook publishers, this resource is nothing short of an object of veneration and is cited to resolve all editorial disputes. It contains rules for every conceivable usage, punctuation, and grammar scenario. It's particularly useful for tricky questions about hyphenation of compound words, phrases, etc. Unfortunately, it's a subscription resource; it's up to you to consider whether you need it that much.
- Strunk and White, The Elements of Style: Another legendary resource, this one has the advantage of being short, but containing most of the advice anyone needs on how to write well. I've linked to the Internet Archive's free PDF copy; I'd also recommend you buy one.

Common editorial issues

I've chosen here to highlight the most common issues I have run across in my years of editing technical content, both here and at previous positions. As you edit, be on the lookout especially for everything discussed in this section.

Writing to persuade rather than inform

When editing technical documentation, you need to be aware of the difference between documentation and marketing: documentation seeks to inform, while marketing seeks to persuade. The distinction between these purposes manifests itself in the type of statements that documentation and marketing copy make:

- Documentation makes statements of *fact* (specifications, descriptions, definitions; what things are and how they work) and *instructions* (do this, then do that).
- Marketing makes statements of *opinion and emotion* (value judgements, comparisons) and *calls to action*.

In most cases, the context of a given piece of writing is clear: the substantial content of the Developer Center is technical documentation, while we have a layer of what we call "marketing pages" that essentially tell our audience how great our technology is and encourage them to use it. However, it's easy to allow the language of marketing to creep into technical documentation, and writers often do so without being consciously aware of it. As an editor, you need to be aware of the distinction between documentation and marketing and make sure that documentation you edit doesn't contain marketing language.

Missing context/background info

No instructional writing can serve its purpose if it assumes the audience knows things that they actually don't. See Understanding the reader for further discussion of what we assume about our audience. Be aware of issues in a piece caused by the failure to identify concepts and terminology, most especially those specific to Docusign, and either remedy those issues (usually it's as simple as cross-linking a term or adding a brief defining phrase) or give the author feedback on what they need to do to fix the issue.

The power and perils of writing from a template or adapting an existing page

Templates make it simpler, easier, and much quicker to develop a new topic; reusing content works! However, when a piece of content has been based on a template, or has reused content from another page, make sure that no material has been left in place that doesn't fit the current topic. For example: if the piece is about an SDK and contains instructions for installing it, make sure those are the correct ones, not the instructions from another SDK page this one is based on.

Inappropriate colloquialism and idiomatic expressions

Collequialism and idiomatic expressions create contextual problems beyond the simple fact that they're informal usage. They generally come from a specific culture or context and are opaque to anyone not familiar with that context. For example, ask anyone in the South what "bless your heart" means!

Idiomatic expressions aren't just informal language; they're nonliteral phrases, generally specific to a country or cultural context, that only make sense if you know the metaphor or analogy they're based on. If you've ever observed that someone "hit the nail on the head", you likely weren't referring to their carpentry skills.

Here are a few more examples:

- Beat around the bush
- Get your act together
- Steal someone's thunder
- Under the weather
- Nail it
- Go the extra mile
- Wake up on the wrong side of the bed

Contextual issues are often as broad as nationality, as the way English is used around the world has innumerable variations. Colloquialism and idiomatic expressions present barriers to understanding for people who either aren't from or don't recognize the context these expressions come from, and are difficult for translators (which, in our case, is basically Google Translate and other automated tools) to translate correctly. When editing, identify colloquialism and idiomatic expressions and always edit them out of our documentation.

Dangling and misplaced modifiers

Dangling or misplaced modifiers are some of the most common problems in sentence construction, and I see new cases every day. It's perhaps easiest to explain via example:

Even in my pajamas, the elephant was an easy target.

So, who's in their pajamas? Per the rules of English grammar, it's the elephant. This is a *dangling* modifier, because the sentence doesn't include the actor associated with the introductory phrase. Silly stuff like this may seem unlikely, but here's a more typical example:

While driving to work, a car accident was witnessed on the highway.

No, the car accident wasn't driving to work, either. This kind of dangling modifier shows up in technical writing all the time, due to the tendency to use passive voice. When you write so as to leave the actor out of the action, you can inadvertently create sentences like this, and as an editor, it's your job to spot and correct them every time; the reader should never be left in doubt about who or what is performing the action in the sentence.

One more subtle example, this one from our own Developer Center:

While not recommended, an app can be used in multiple integrations.

What's not recommended here? The app, apparently. While we can project that a reader would probably understand our intention here, there's an ambiguity we don't need.

Misplaced modifiers can be similar to dangling modifiers, but in this case, the thing being referenced is in the sentence; the modifier is just misplaced in a way that causes ambiguity or confusion. Examples:

We glued together the vase we broke quietly.

No, the vase wasn't broken quietly; we just quietly glued it back together. Notice that here, there's no doubt about who's doing the action, but the sentence still conveys the wrong meaning, simply because the adverb isn't placed next to the verb it's modifying. This kind of error crops up in any kind of writing, including documentation.

Unclear antecedents

This problem crops up a lot in technical writing, partly because of passive voice, but also because we tend to construct overly complex sentences in which it's hard to track *who* or *what* is doing *what* to *whom*. Here's a simple example:

Colleen called Alicia while she was doing her homework.

Who's doing their homework: Colleen or Alicia? It's impossible to tell. When you find instances like this in our writing, you'll need to query the author about what they meant.

More commonly in technical documentation, unclear antecedents occur with *it*, *this*, *that*, and *which*. Consider this example:

The Westville Chamber of Commerce has a dynamic local business network, and it can help your restaurant succeed in a crowded restaurant market. It helps business owners collectively solve problems and implement solutions.

There are two instances of *it* here, and it's not clear what either one of them refers to. (Hey, notice the example of ending a sentence with a preposition! This is an example of a good time to do so, as the alternative is "…it's not clear to which either one of them refers", which is a bit formal.)

Overly complex sentence structure

Complex sentences happen a lot in technical writing, caused often by passive voice, the desire to include everything at once, or the fact that you're trying to explain a situation with several actors at once. This is one of those things that's easy to recognize and fix with a general rule:

Never allow a sentence to have more than two clauses, or one introductory phrase plus an independent clause.

If you see more than two coordinating conjunctions in a sentence, or more than two semicolons or colons, beware. There are probably more than two clauses in the sentence. Break it into two sentences, or even more depending on how complex it is, by replacing one or more of the conjunctions or semicolons with a period.

Lack of parallel construction

Parallel construction means to use the same grammatical forms, patterns, or tenses. This helps make writing clear and powerful by bringing rhythm, flow, balance, and emphasis to a passage.

Parallel construction is commonly used at the level of single words, phrases, and clauses:

- **Words:** I like reading, traveling, and playing sports.
- **Phrases:** …and that government of the people, by the people, for the people, shall not perish from the earth. —Abraham Lincoln
- **Clauses:** The inherent vice of capitalism is the unequal sharing of blessings; the inherent virtue of socialism is the equal sharing of miseries. —Winston Churchill

In technical writing, we don't often create faulty parallelism at the word level, but it does happen at the phrase and clause level. Often it's the result of using different parts of speech:

Example: To make calls to the eSignature REST API, you need three things: a developer account, an integration key, and obtaining an access token. (By the way: Grammarly saw nothing wrong with this sentence.)

Correction: To make calls to the eSignature REST API, you need to obtain three things: a developer account, an integration key, and an access token.

Or casting one phrase or clause in passive voice:

Example: While we used to endorse Implicit Grant authentication, Authorization Code Grant with PKCE is now preferred.

Correction: While we used to endorse Implicit Grant authentication, we now recommend Authorization Code Grant with PKCE.

Sometimes it becomes a source of actual confusion:

Example: *My ADHD is worse than my wife.* (Really? How do you make that comparison?)

Correction: My ADHD is worse than my wife's. (Of course!)

Passive voice

Passive voice is making the subject of the sentence the receiver of the action, rather than the actor performing that action. This is a weakness in any writing, as passive voice lacks energy; however, in technical writing, it often creates actual confusion, because when the subject receives the action of the verb, what's left unsaid is who or what is actually performing that action! This leaves the reader in doubt of who, or what, is doing things, which can make it difficult to understand what we're trying to say.

Example: It is recommended to use exactly one integration key (and one app) per integration. (By whom? Us, industry experts, Docusign?)

Correction: Docusign recommends using exactly one integration key (and one app) per integration. (This is how it's expressed on the site, by the way.)

We used to address the author and reader

You see *we* used a lot in technical documentation, blog posts, etc. Writers commonly adopt this way of addressing the reader; to most of us, it comes naturally. It's based on a fallacy, however: in no case are *we* in this sense doing anything together. The writer is either describing actions they are taking themselves, or they're instructing the reader to do something; so it's not *we,* it's *I* or *you*. When presenting instructions, it's best to address the reader directly as *you:*

To make the API call shown in this how-to, you need a valid OAuth access token.

or, using the imperative, simply leave out *you:*

Create a new, signable HTML document in code.

There are acceptable uses of *we* in our writing, however:

- In a credited piece such as a blog post that has more than one author, *we* can refer to the authors.
- In any context, *we* can refer to Docusign as a company, or any subgroup within Docusign as a unit.
- *We* can also refer generally to the collective body of people in our context: developers, people in high tech, or any collective body of people that makes sense.

Misuse of dashes vs. colons

Colons and dashes are often used interchangeably; that is, a lot of writers use dashes when they should use colons. Rather than lay out all the uses of each, I'm going to refer to a good blog on Grammarly:

Semicolons vs. Colons vs. Dashes

Be alert for this issue and replace a dash where a colon should be used.

Overuse of exclamation points

This is a problem that more often occurs in blog posts, community posts, and marketing communications. We rarely, if ever, use exclamation points in our documentation; but in blog posts, community posts, and marketing communications, we often want to express some excitement over what we're talking about. These are fine in moderation, but beware of excess use and edit out overused exclamation points. This is a judgment call, but as a rule, if everything's exciting, nothing is.

Oxford comma; commas in general

There are definite rules for when to use a comma. If you've been told "just put one in wherever you'd pause while reading the sentence," I'm sorry, *no*. Learn the rules of comma usage so you can recognize how to add or remove them properly. Grammarly.com has a good short guide to comma use.

One especially important misuse of commas is the comma splice, in which a comma is incorrectly used to join two independent clauses. Beware of this error and fix it with either a semicolon, a conjunction, or making the clauses two sentences.

About the Oxford comma: This is worth pointing out, because it's the one area where DevCAD (and ProdDocs) differ from corporate style. **We use it** because it's often necessary to make our meaning clear; **Docusign corporate communications does not,** because their style is based on the AP Stylebook (originally developed for journalists), which works well for marketing communications.

Unnecessary hyphens: Prefixes, phrases after the verb

Technical writers tend to hyphenate a lot, use hyphens where they're not needed, and use them in place of other punctuation. There are lots of terms in software development that you won't find in the dictionary, and it's tempting (and often done) in documentation to hyphenate terms that are used mostly in high tech, and are sometimes even coined for the purpose. Bear in mind the following general guidelines:

- Check the dictionary first. If it's spelled closed in the dictionary, don't hyphenate.
- Most words formed with prefixes (*pre-, multi-, non-, re-, macro-, semi-, sub-,* etc.) don't need a hyphen. Even if they're not explicitly listed in the dictionary, there's no need to hyphenate the vast majority of such terms. Do check the dictionary, however, because there are exceptions to this rule; and in some cases, a hyphen may be required for clarity (*re-creation* vs. *recreation*, for instance).

- When a phrase is used as an adjective, it's most often hyphenated when it appears before the noun it modifies, but spelled open when it's after the verb. For example:
  These "real-world" examples don't apply well in the real world.
- Don't use hyphens in place of other punctuation. Two hyphens are not a dash; that's a convention left over from the days of typewriters. The sequence dash-hyphen-dash (-) is never correct. Numerical ranges are denoted with en dashes; when you think of using a dash in most of the ways we do, you're thinking of an em dash.

  Note: To access dashes in Google Docs, select Insert > Symbols > Special Characters; then choose Punctuation from the first dropdown and Dash/Connector from the second dropdown.

Approach to the work

So what do you do when you're asked to perform an editorial review on someone else's work? Before you begin making tweaks to the copy, you need to get your bearings first. Ask yourself:

- Do I understand the subject?
- Where can I get information if I need to learn about it?
- Is this content modeled on a template or another page on our site?
- What's the purpose of this content: to inform, to instruct, or to persuade?

You need to have clear answers to these questions before you can begin to edit the content.

One pass vs. multiple

Is it easier to make one pass through a document, correcting all editorial issues you see as you go? Or would it be more efficient, or result in better quality, to make multiple passes to tackle different issues? This is something you can decide for yourself, and the answer to this question speaks most to your own strengths as an editor: your eye for detail, your understanding of the material, and simply your ability to focus.

It's common among editors, and good practice, to simply give the copy a good read first, without making any attempt at editing at all. This helps you understand the content, makes you aware of its structure and purpose, and gives you context for understanding the author's intent and why they made the choices they did while writing. It's probably okay to spot-fix small issues as you see them, such as punctuation, extra spaces, and formatting; but, if you do, beware that some of those edits may need to be reverted once you've had a full read.

Once you've read the content, you're ready to make a more substantial judgment of its overall quality and structure and make edits to address any such concerns. Tackle larger issues first, as they'll have ripple effects through the work that will determine the need for other, smaller edits.

Usually, as you do this, you'll find places where substantive edits are required that may affect entire paragraphs or sections of the document.

During this process, you may find it expedient to also take care of any little edits that you notice the need for as you go. However, if the work needed to fix larger issues requires substantive rewriting or changes in organization, it's probably better to do smaller edits in a subsequent pass that lets you focus on wording changes that are the consequence of the larger edits you've already done.

Finally, some editors reserve a last pass to look for individual typos, punctuation errors, extra spaces, and formatting issues that they may have missed while doing the more substantive edits, as well as to catch any inconsistencies in their own work.

Chunks vs. full pass

Most of the content we create is small enough (a single web page) that you can easily do it in a single go, the whole document at once. However, in some cases, changes to multiple pages may be the same doc, along with a draft of an entirely new page. This is a good thing, because you can review related material across many pages at the same time; but depending on how big the document is, and how much time you've got, you may find it expedient to work on part of the document at a time. When you do this, it's good practice to add a comment to yourself to mark the spot where you stopped working so you can pick up from there later.

Search and replace: Power and perils

Search and replace seems like a good thing, and when properly used, it can be a great saver of labor and a tool for enforcing consistency. However, when using it in the context of technical documentation, beware: exceptions to whatever you're replacing abound, and using a global search-and-replace operation to edit technical content runs the substantial risk of changing meaning and introducing error, which are the cardinal sins of technical editing.

One reason for this is that the same string may be used to refer to different things in different contexts. For the sake of one context, the string may need to be changed, while for another context, it needs to remain the same. Another problem is that our content often contains code, and within that context, changing the string will make the code invalid.

With that possibility in mind, whenever you use search and replace, do NOT replace all at once; go from one found instance to the next, and individually evaluate each instance to make sure it's appropriate to make the replacement

Using assistive tools: Spelling checkers, Grammarly, AI

Spelling checkers and grammar tools such as Grammarly work great while you're writing, but I've never found them much use in editing. You can presume that the author used such tools while

writing as well, so anything those tools can fix has already been fixed. These tools also don't address common problems:

- They can't deal with words not in their dictionaries, so they often incorrectly flag words that are actually correct in context.
- They don't recognize words used incorrectly. Many times, a word is spelled correctly, but it's simply not the correct word to use.
- They assume anything hyphenated is correct, if the words making up the hyphenated compound are themselves spelled correctly.
- Grammar tools often misunderstand the intended meaning, so they make suggestions that work, but don't mean what the writer wants to say.

For all these reasons, I don't pay much attention to spelling and grammar tools while editing. If such a tool has flagged a word or passage in the text you're editing, check its recommendation against your own judgment.

What about AI? I have (not deeply) investigated AI tools for editing purposes. From what I've seen, there are some nifty ones out there; they even track changes for you. I see the following difficulties, however:

- Depending on the way you prompt the AI to perform the edit, its edits may be trivial, no better than spelling and grammar checkers; or way too heavy, trampling on the author's tone and voice.
- To edit our content effectively, you have to understand, very specifically, the context of the topic. AI, which is trained on a broad base of information, doesn't have the specific domain knowledge of our technology to edit without introducing inaccuracy.
- AI doesn't understand the specific content models we work with, so can make changes that violate those models. It's also likely to change boilerplate language that we don't want to change at all.
- There's no guarantee that AI will understand the author's intended meaning, and therefore no guarantee that AI won't inadvertently change it.
- While AI editors track changes, the ones I've tested don't work directly in Google Docs, and if you paste the AI editor's work back into the draft doc, those changes don't get tracked correctly. Instead, you wind up with a document containing both deleted and inserted text, with no changes tracked at all. This obviates the entire point of the exercise, which is to save the editor's and author's time.
- In making edits, AI is drawing from its knowledge base, which almost certainly includes copyrighted material; so in making edits, AI can introduce phrases, or even whole sentences, lifted from a copyrighted source. Using this would make us guilty of plagiarism and copyright infringement, thus making our own material uncopyrightable. There are corresponding issues with using AI to code, which is why Docusign has a policy forbidding its use in creating production code for our products.

For all these reasons, I've not used AI editing tools. I don't see this as a solution for editing our content.

Editorial workflow at DevCAD

It's important to understand the way editorial review works at DevCAD so you can edit appropriately and work with your colleagues productively. Here's an overview of the process.

Draft

From an editor's point of view, the draft phase is a black box; we don't see the document until the author feels it's ready for editorial review. One thing we do insist on, however: no document is ready for editorial review until after it's been technically reviewed, and the results of the technical review have been implemented to the satisfaction of the reviewer. There's absolutely no point in editing work that may substantially change for reasons of technical accuracy.

Review

Review is the editor's pass at the document. The author should set sharing permissions on the Google doc to assign everyone at Docusign the Commenter role; this allows tracked changes as well. Make sure, if the author shares a document with you and assigns you the Editor role, that you work in Suggesting mode; if you work in Editing mode, your changes will not be tracked.

During editing, the editor may, and should, query the author via comments about anything in the document for which they have concerns or questions. This sets up a dialogue between the two, which should be resolved on all points before editing can be considered complete.

Resolve

Once the review is complete, the editor notifies the author and the author takes the responsibility of resolving all the tracked changes. For new content, assuming both parties have executed their roles with respect for each other's work, this should be as simple as accepting all changes and removing all comments. For revisions to existing content, the author should review and approve edits, indicating such with a separate communication to the editor, but leaving the tracked changes in place; this enables those changes to be more easily implemented on the corresponding web page.

Production

The document is ready for production in all edits and comments are resolved. Usually, the author produces their own work in Contentful; however, depending on bandwidth or project assignments, another person may be tasked with production.

Review

Before publishing, the produced document needs to be checked against the final Google doc to make sure that all the content has been built and renders as intended; the producer is responsible for fixing any issues identified during review.

Publish

Once the produced document is signed off on preview, whoever owns the corresponding Jira ticket must update its status to “In Validation” and assign the current fix version. Once the content is released, its ticket can be closed.

Resources

Consider the following resources to inform your editorial work:

- Developer Content Style Guide
- API Documentation Style Guide
- Docusign Copy Style Guide
- Microsoft Writing Style Guide
- Merriam-Webster Online
- Grammarly Blog
- Chicago Manual of Style Online
- Strunk and White, The Elements of Style