How to write blog posts for web.dev and developer.chrome.com

- Getting started
  - Does this post live on web.dev or developer.chrome.com? {: #which-site}
- Writing style guide
- How to format your draft
- The general procedure
- Key writing tips
  - Article length
- Working with an editor
- Tips for writing technical articles
  - Define the target reader
  - Write an outline
  - Write a concise introduction
  - Create good code examples
  - Share knowledge that's not in the docs
  - Think about different ways to share information with components
  - Give credit and avoid plagiarism

This guide is written for people who are not primarily tech writers, but want to write a post for web.dev or developer.chrome.com. It links to some of our key writing information, explains our process, and provides tips for writing technical blog posts.

Getting started

The Quick Start Guide is an overview of proposing your new content. The content form asks you whether the post should be posted to web.dev or developer.chrome.com.

Does this post live on web.dev or developer.chrome.com? {: #which-site}

The two sites serve different audiences. If your post is about a web platform feature that is not Chrome specific, and is implemented, or in the process of being implemented in other browsers, then it can be posted to web.dev.

If the post is about Chrome-only features that do not yet have broad consensus on implementation, or about the browser engine itself, it belongs on developer.chrome.com. If unsure, mention this in your content proposal and we can help you work it out.

Writing style guide

The writing style guide hosted on web.dev is your reference for style, voice, and tone across both sites. There are some technical differences in terms of components you have access to when formatting your post:

- web.dev components
- developer.chrome.com components

How to format your draft

Use Google docs initially, and write in Markdown. Make a copy of this template as a starting point. Your initial reviews will take place in the document as it is an easier place for structural changes than GitHub. You can also share your document with any members of your team who need to sign off on the content before publication.

The general procedure

A good general order of operations looks like this:

1. Write your outline.
2. Get technical writer feedback on your structure.

3. Draft your article.
4. Get a content review from technical experts or other interested parties.
5. Revise your article as needed.
6. Get a copy edit for typos, grammar issues, and other needed corrections from the technical writer.

This order helps you avoid:

- Multiple content reviews because you changed the structure.
- Multiple copy edits because your content reviewers requested wording changes.

Expect each review step to take as much as a week. All parties are juggling competing priorities.

Key writing tips

We don't expect you to memorize the style guide. Following a few key points will really help, and your assigned tech writer will help with the rest.

- Write posts in a voice that is knowledgeable, humble, sympathetic, and friendly.
- Use simple language, the present tense, and the second person (you, rather than we). The grammar page of the handbook has some useful examples.
- Assume readers understand basic web development in HTML, CSS, and JavaScript.
- Use the content checklists as a reference. They give you an idea of what the editor will be looking for.

Article length

A good length for posts on web.dev is around 1000 to 1500 words. If you find that you are writing much more than that, it might be that the article can be split into two or more pieces dealing with various parts of the topic. Ask your editor for advice on this. Also, see the tips below and consider whether you are over-explaining basic concepts.

Working with an editor

Every post will have a tech writer assigned. They will act as your editor, and help to make sure your post fits the audience for the site, and that style guide details are followed.

Working with a tech writer means that you don't need to worry about every detail of the style guide. It also means, however, that your carefully crafted post might come back covered in suggestions and comments. This can be a bit of a shock the first time it happens. A document

covered in suggestions doesn't mean that you did a bad job. The majority of these changes will be copy-edits to follow the style guide and you can approve those and move on.

You will generally be assigned a tech writer familiar with your subject matter, and they may well spot places where the post could be more clear, however you are the expert. Always query any change that you feel is incorrect, or unhelpful. These discussions between editor and writer very often lead to changes that improve the clarity of the post.

Tips for writing technical articles

The following suggestions may be helpful when working on your post.

Define the target reader

Defining who your post is for can help you to avoid two common issues with technical articles:

1. The article explains every concept in detail as if the reader has never heard of anything mentioned.
2. The article mentions new technical terms or concepts with no explanation at all.

The correct level of detail is usually somewhere between these places, and to know exactly where your line will fall, you need to define who is reading the piece.

Most web.dev posts can assume that the reader understands basic HTML, CSS, and JavaScript. With that as a baseline, ask yourself whether your article requires any other pre-existing knowledge. Do you intend to cover this knowledge in the article or consider it a prerequisite?

As you write, you can bring this ideal reader to mind when deciding whether to fully explain a feature, add a link, or assume it is understood.

Write an outline

Putting together an outline for your post is a good first step. It helps you to organize your thoughts on the subject. An outline also helps you to see how much material you have to cover.

Posts on web.dev tend to be short. If you have a lot of material to cover, consider how to break it down. You could write an overview, then write more detailed pieces on specific parts of the feature later. Or, it might be that there is already a good introduction to the subject available on the web; therefore, your post could go into more detail on a particular part of the feature and link to the third-party overview.

An outline doesn't need to be anything fancy. A list of the main headings with a brief description of what you will cover in each section is all it takes.

If you share this outline with the tech writer working with you, they can make suggestions and help you to decide what to cover. Remember to let them know who you have identified as the target audience for this piece.

Write a concise introduction

The introduction lets your target reader know that this article is for them and what they will learn by reading it. It's your chance to grab people as they scroll through the articles looking for something to read. Introductions should include:

- Who is this for?
- What do they already need to know?
- What will they learn?

Make your introduction as concise as possible. When there are prerequisites for knowledge (over and above HTML, CSS, and JavaScript), link to a good resource.

Create good code examples

When creating a code example, ask the question, "How can I demonstrate this feature in the simplest possible way?" Good code examples have a lot in common with the reduced test case. They rarely resemble production code.

For inline code examples, check out the tips in the handbook.

In CodePen or Glitch examples, avoid using languages that compile to HTML, CSS, or JavaScript (unless the post is about those alternate languages). Remember our ideal reader, who we know understands HTML, CSS, and JavaScript? Don't add a barrier to understanding your demo.

Try to avoid large amounts of additional JavaScript or CSS styling. Making your demo look pretty isn't the most important thing.

Ensure that any dependencies have at least the same browser support as the thing you are demonstrating. Otherwise the reader may be confused as to why the demo doesn't work. If you're writing about a new feature, state which browsers it currently works on in both the article and the demo.

If you want to include a fully worked demo, include that as a link and use it as a chance to show off the feature.

While reducing things to the minimum needed is always the goal, take care that this doesn't leave a copy and pasteable example that misses something important to making the code secure. In particular, avoid creating live examples on CodePen or Glitch with security or similar issues. You do not know where they might end up.

If you want to avoid writing validation or other code, adding the example as inline code with a comment indicating the position of the required additional code makes the intention clear.

Share knowledge that's not in the docs

Hopefully, there is already a reference on MDN or elsewhere for your feature. The aim of your post should not be to write an exhaustive reference but to give extra information and best practices. That might be in the format of sharing cool use cases, providing tips from one developer to another, or explaining how some features work together. Suppose you find yourself detailing each method of an interface or every value of a CSS property. In that case, you are writing reference and not an article, and there are better venues for docs.

Instead, write about what these methods or values can do, show examples, then link to the docs where the reader can explore how to take those examples further.

If no reference is available, but you think there should be, contact jmedley@ who can assist you in creating it.

Think about different ways to share information with components

On web.dev there are several interesting components. Using these in your article will avoid presenting readers with a wall of text. For example:

- Add extra information or warn people with the aside component: aside on web.dev, aside on developer.chrome.com.
- Contrast code examples with the compare component: compare on web.dev, compare on developer.chrome.com.
- Add browser compatibility information on web.dev.

Give credit and avoid plagiarism

If your thinking around a subject is informed by other work, reference it. While directly copying is well understood as plagiarism, technical articles often indirectly draw on other people's work. Perhaps you watched a conference talk or video, read some blog posts, and even asked questions

in a forum where someone gave you help. Mentioning these sources strengthens your article. Referencing shows that you have done your research, gives readers the chance to find out more from other people, and ensures that other folks get credit for their work.

If you directly quote or paraphrase someone, then include a link in the text. If there is not a good place to add a link inline, add the piece to a resources or credits section at the end of the article.

□□□ #3
□ □□□ □□□ 2026-10-31 22:41:52 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:59 UTC