swimm

Writing an effective code document in 5 steps 🎯

swimm.io

Introduction

This guide will teach you how to write **effective** documentation, and how to do so easily with Swimm.

> **Effective documentation helps other developers, technical staff and users achieve their goals.**

Our experience stems from documenting ourselves, and accompanying many engineers creating effective documentation. We have seen again and again how following a few guidelines can make a huge difference - both for creating more effective documentation and making the process easier for the writer.

How to write an effective document

This section assumes you know what document you want to write. If you are unsure, refer to the specific use cases section.

Writing an effective code document in 5 steps 🎯

This flow is easy to follow, and already gets you far ahead in creating effective documentation.

1. **Give your document an actionable title**

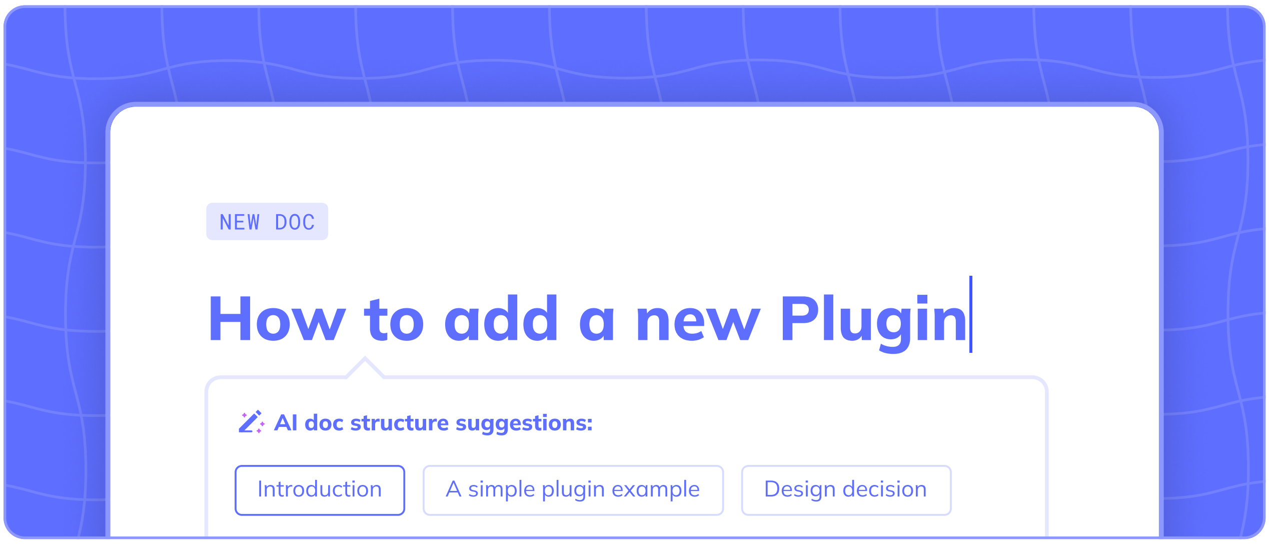
   Don't skip this step. It will help you focus.
   When it makes sense, use one of these formats:

   → "How to..." (e.g., "How to add a new Plugin")

   → "How X works"(e.g., "How the recommendations engine works")

   → "How we built X" (e.g., "How we built our CLI")

   The title should tell the reader **what they will learn** from reading this doc.



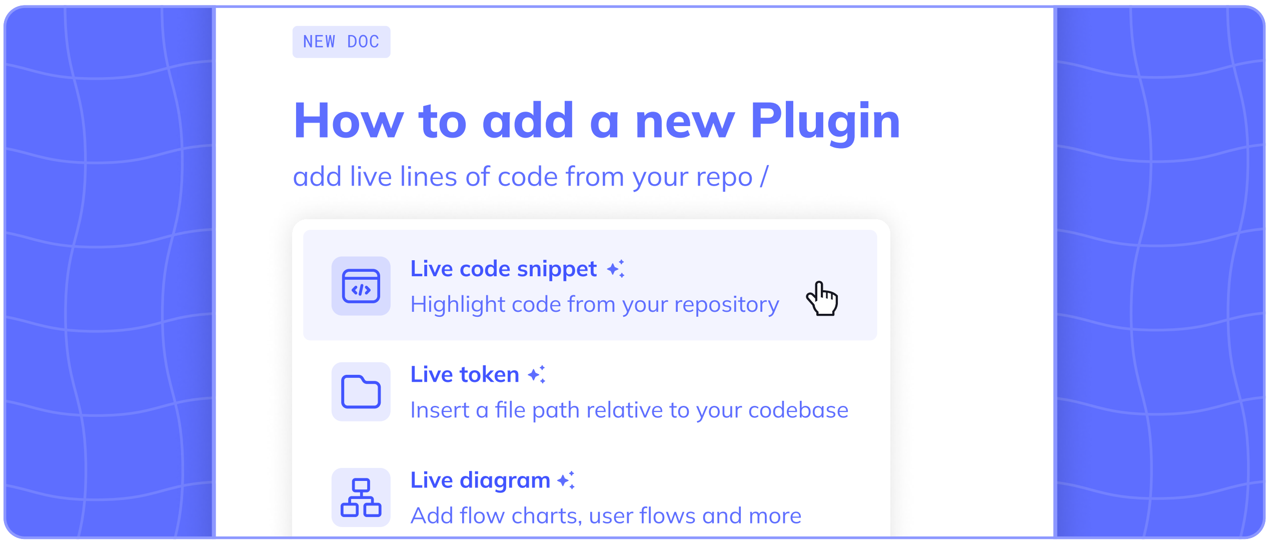
swim

How to write an effective document

2. **Select code snippets first, before writing any text**

→ Use the Swimm command `/Code snippet` Before explaining why your code is important, focus on adding all code snippets that show the flow you are describing. If you are not describing a flow, select an example that demonstrates what you are describing, or any code snippet that might be relevant. If you are unsure what snippets you should add, it might be helpful to think of someone who doesn't know the code and highlight what you would show if you walked them through it.

→ Make sure you add all relevant snippets.


NEW DOC

How to add a new Plugin

add live lines of code from your repo /

Live code snippet
Highlight code from your repository
Live token
Insert a file path relative to your codebase
Live diagram
Add flow charts, user flows and more

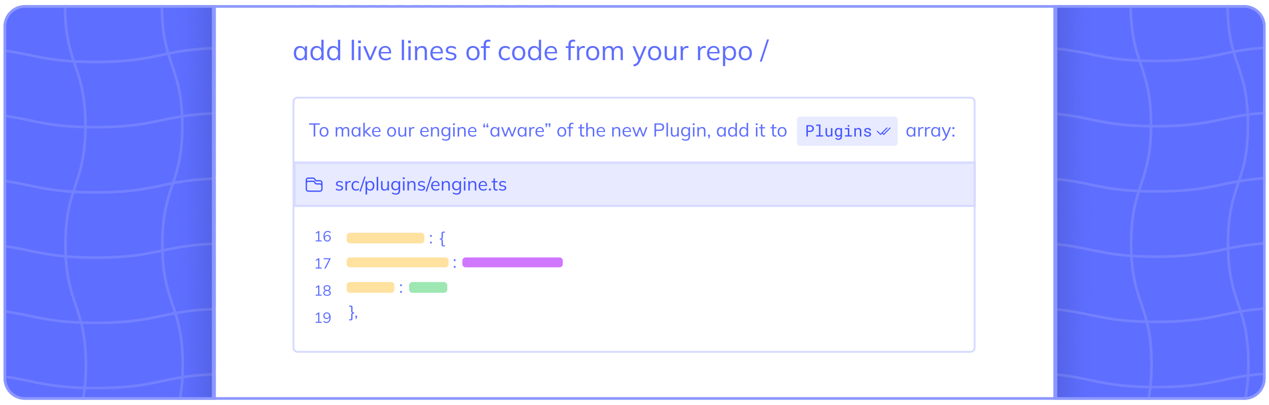
How to write an effective document

3. **Describe the snippets**

   Now that you have your snippets, you can re-order them (if necessary), and describe them. **Pro tips:**

   - Explain **why** things are implemented the way they are.
   - Focus on the information that's **not in the code**.
   - Explain **how** a single snippet relates to the other snippets - its role in the flow.
   - Refrain from explaining **what** exactly each line of code does. The code speaks for itself in isolation, and can tell parts of the story for you.
   - Use **smart tokens** that are coupled to elements from the snippet or other locations in the code. Type backtick `'` to search for code references in your repo and convert them to tokens.

   The title should tell the reader **what they will learn** from reading this doc.



add live lines of code from your repo /

To make our engine "aware" of the new Plugin, add it to `Plugins` array:

src/plugins/engine.ts

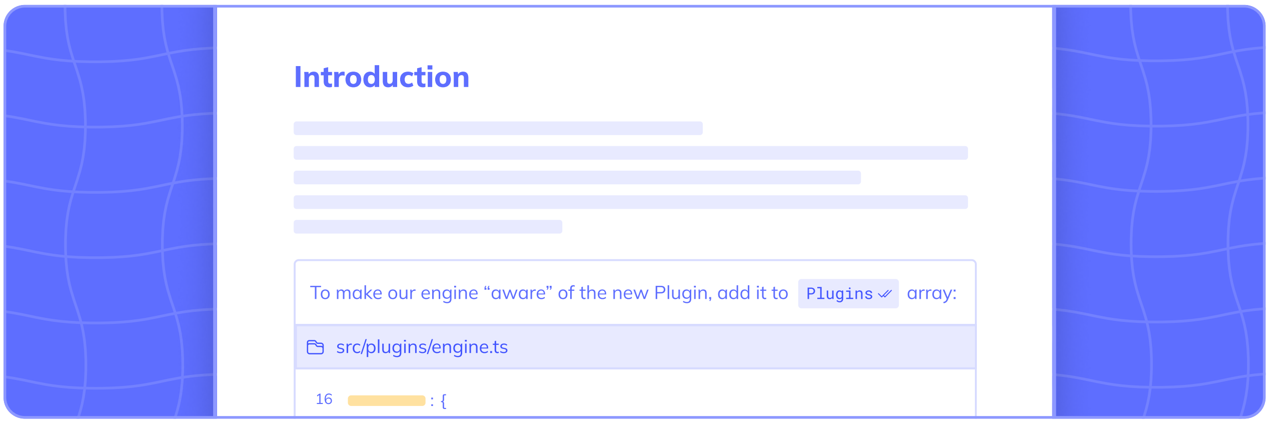
```
: {
:
:
},
```

swim

How to write an effective document

4. Add an Introduction section

Explain what this doc is about in an introduction section.



Introduction

To make our engine "aware" of the new Plugin, add it to `Plugins` array:

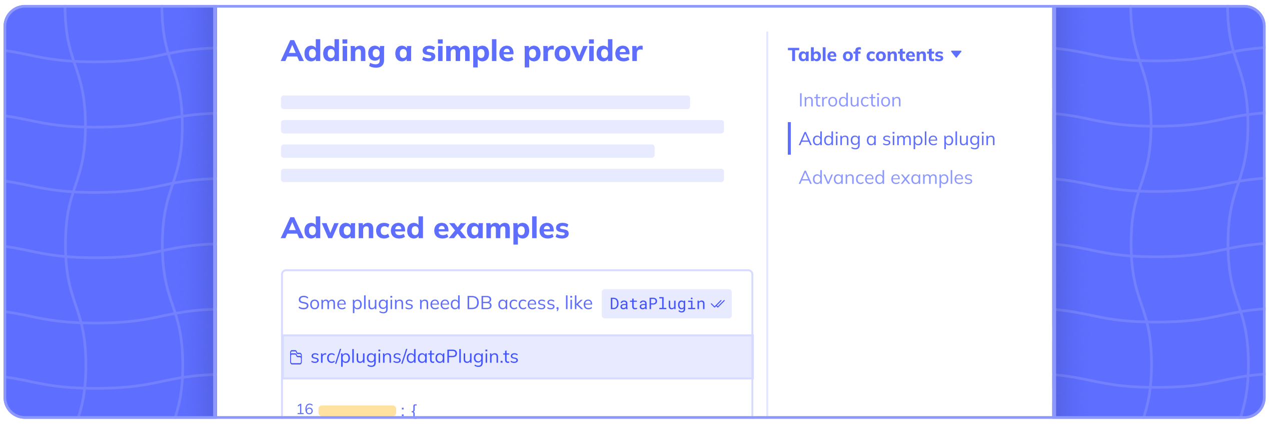
src/plugins/engine.ts

```
: {
```

5. When applicable, split into sections

Create sections using headings. They make the document easier to follow and navigate.

→ In the Swimm web app, a Table of Contents navigation is automatically created on your right sidebar based on Markdown headings. Review it to make sure your structure is clear.



Adding a simple provider

Advanced examples

Some plugins need DB access, like `DataPlugin`

src/plugins/dataPlugin.ts

```
: {
```

swim

Why are these steps so effective?

- It helps avoid **writer's block**. By focusing first on selecting code snippets, you don't mind your brain with copy or styling choices. When you get to actual writing, you'll already have a structure and code to prompt you, which is much better than a blank page.
- It makes you select a real example.
- The **code speaks for itself**, at least in isolation. By including these parts of the code, you don't need to explain them in English. You can then focus on parts of the story that are not clear within the code itself.

Why a real example?

1. **It's easy to understand.** As a developer, a real example is easy to relate to. It also provides a good basis to rely on when the developer would look to use the tool themselves.

2. **It's easy to create such a document.** If an example already exists in your codebase, there's no need to invent a new one. All you have to do is describe it.

3. **It helps you remember.** When you look at a concrete, real example - you see all the small implementation details. Not all of them are important to mention or explain, yet it makes sure you don't forget about those that are.

4. **It's maintainable.** By code coupling to an existing example, if something ever changes in the system and the example changes, your document will be updated.

4. **It's easy to discover.** Thanks to the discoverability of Swimm documents, they are found when someone uses this tool. For example, when using a library for the first time, a developer may look for other usages of this library in the codebase, such as the wrapper function. Thanks to Swimm's IDE plugins, developers are likely to find the relevant document next to the wrapper function because it was referenced and code-coupled in the document.

4. **It’s the *right* example.** When writing new code, developers often look for previous examples in the code to copy or learn from. Some of these examples are sub-optimal, don’t follow best practices or new conventions or just are not a good fit for the use case you’re describing. Adding the right example from the code helps to avoid the perpetuation of errors and “bad habits” in your codebase.

Pro tips

- When describing code, use the code as part of the explanation (with snippets, tokens, and paths). This serves both to make the explanation more concise and explicit as well as make sure it remains up to date.
- Add tags, and add the doc to relevant playlists and folders.
- Consider adding a Mermaid diagram. Code-couple your diagram with smart tokens!
- Consider creating doc rules to help people find this document when it's most relevant.

Use cases

We maintain a section on our documentation site with descriptions of various use cases and how to approach them. Some of these use cases are:

- **Increase your bus factor**
- **Support an infrastructure change**
- **Explore and document legacy code**
- **Onboard new developers**
- **Promote a new tool**
- **Promote testing best practices**

Please check out this section on our documentation site.

Have questions?
Please reach out to the Swimm team!