Trying to convince application developers to write API documentation

Sébastien Wilmet

January 3, 2020

Contents

Abstract

This article tries to convince application developers to write API documentation. We first look at libraries, and what are the requisite ingredients to make the programmer's life easier (note, not all libraries or development platforms follow all those ingredients, so there is room for improvement in libraries as well). Then we look at applications, why an application usually lacks proper API documentation, and what problems it causes.

1 Working with libraries

A library – if well developed – has documentation to use its API, with the possibility to read the documentation without opening the source code. If you have luck, the documentation can be browsed in an API browser, with an advanced search functionality to find symbols of the library. And if you have even more luck, the API browser is linked with your text editor, to directly open the documentation of the symbol under the cursor from the text editor. Actually this should be a minimum to have when programming and using libraries.

Don't forget what does the "I" of "API" mean. API is the acronym for Application Programming Interface. The *interface* is the external surface of the library, what users need to know about it in order to use it. If properly done, the interface hides the implementation details (it's called information hiding, or encapsulation). For the library developer, it means that everything that is *not* part of the API can be changed without incrementing the major version number of the library, to keep backward compatibility. Keeping certain details hidden gives more freedom for the library developer. And

library users hate when there are API breaks. So it's not a good solution to make everything public, there needs to be a design balance between what is made public and what not.

An API without documentation is usually not an API. Just providing the functions' signatures (parameter types and return value type) is often not sufficient to know how to use the functions. But in a lot of simple cases, the function signature *is* enough, with the names and the types it's self-evident. So the library developer may be tempted to write documentation only for the functions that require it. The problem is the following: for the undocumented functions, can their signatures be extracted from the source code, to show it in the API browser? If not, then the library users need to read the source code, and this can lead to other problems (read next paragraph).

If you need to open the source code of a library to read its API, there is the risk to read its implementation and see implementation details. Then the user of the library will be tempted to take advantage of the implementation details. This can lead to bad situations. If the library developer change that implementation detail, it will break the application. Then the application developer will be angry against the library developer. Or when the library developer knows in advance that if he or she changes that implementation detail it will break certain applications, then he or she may decide to not make the change, which brings the library developer with less freedom (or the need to create a new major version).

All in all, the user of a library is happy when the documentation is available in an API browser, or at least when there is no need to open the library source code. If the documentation is well structured, for example with related classes grouped together, it makes the library easy to learn. When needing to open the source code, there are way too much information to have a gist of the library API as a user, it takes more time to find the information, such a library is thus harder to learn and to work with.

2 Working on an application

We have just seen that using a library can be relatively easy, if it is well done there is no need to open the source code, and the information can be found quickly thanks to the help of an API browser application.

When developing an application, the developer should be aware that it's *also* possible to write API documentation, with that documentation available in an API browser. I think it is safe to say that most applications are not written like this, except the applications that provide a plugin system. In other words, for most applications the developer always needs to open the source code of a class in order to know how to use that class.

For an application, if there is the need to always open the source code of

a class in order to know how to use it, this leads to the following problems:

- It takes more time to find the information, there is too much information.
- For a newcomer who has never worked on the application before, it takes much more time to get accustomed with the codebase.

When a new application is written, when the project is young, the codebase is still small and the developer knows very well the code, so the developer doesn't feel the urge to write API documentation. Then it just continues like that, the codebase grows and grows, documentation comments are added to certain functions that need it, but that's it, the developers work by opening the source code, not by reading the API of a class in an API browser.

So, to whoever reads this, I recommend this thing: write API documentation comments in an application codebase as well, not just for libraries. It will make your life easier in the future, and lower the barrier to entry for newcomers. And it's a good way to manage complexity in a software system.