




# CMinx: A CMake Documentation Generator

Branden Butler<sup>1,2\*</sup> and Ryan M. Richard<sup>1,2\*</sup> 

<sup>1</sup> Ames National Laboratory, Ames, IA, USA <sup>2</sup> Iowa State University, Ames, IA, USA ¶ Corresponding author \* These authors contributed equally.

DOI: [10.21105/joss.04680](https://doi.org/10.21105/joss.04680)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

---

Editor: [Daniel S. Katz](#)  

## Reviewers:

- [@robertodr](#)
- [@peanutfun](#)

Submitted: 12 August 2022

Published: 27 September 2022

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

This manuscript introduces CMinx, a program for generating application programming interface (API) documentation written in the CMake language, and CMake modules in particular. Since most of CMinx’s intended audience is comprised of C/C++ developers, CMinx is designed to operate similar to Doxygen ([Heesch, 2022](#)), the *de facto* C/C++ API documentation tool. Specifically, developers annotate their CMake source with “documentation” comments, which are traditional CMake block comments starting with an extra “[” character. The documentation comments, written in reST ([reStructuredText, 2022](#)), describe to the reader how the functions, parameters, and variables should be used. Running CMinx on the annotated source code generates reST files containing the API documentation. The reST files can then be converted into static websites with tools such as Sphinx ([Sphinx, 2022](#)) or easily converted to another format via Pandoc ([Pandoc, 2022](#)).

Unlike other solutions for documenting CMake modules, CMinx knows the CMake language’s grammar. This enables CMinx to automatically extract function/macro signatures, even when functions are not documented. CMinx also integrates seamlessly into existing CMake build systems. CMinx’s output is highly customizable and easily controlled via a YAML ([YAML, 2022](#)) configuration file. CMinx has already proved to be an invaluable productivity tool in the authors’ other projects, and, given that other scientific software projects also rely heavily on source code written in CMake, we anticipate CMinx will prove invaluable to many additional projects as well.

Concurrent with the submission of this manuscript, we have also released the first production version of CMinx, version 1.0.0. CMinx can be obtained from the Python Packaging Index via “`pip install CMinx`”. Alternatively, CMinx can be used as a CMake module via CMake’s “FetchContent” command. Despite only just releasing 1.0.0, the CMinx GitHub organization has already started to see attention and interest from developers not affiliated with the authors. We anticipate CMinx will be a useful productivity tool for the large swath of research software that uses CMake as their build system.

## Statement of need

The process of building a software package written in a compiled language (e.g., C, C++, Fortran) has always been complicated. Over the years, various build system solutions have evolved to ease the process. Historically, there has been a propensity to treat each build system as a one-off use case. This is understandable since build systems have tended to be relatively small and tightly coupled to the structure and purpose of the package. With build system complexity at an all-time high ([Bartlett et al., 2017](#); [Snir et al., 2014](#)), there is an increasing need to treat the underlying build system infrastructure as code. This means that the build system should be modularized, and those modules should be documented, tested, and reusable. With the popularity of C/C++ for high-performance computing, “build system” is increasingly

becoming synonymous with CMake: there is a desperate need for a robust CMake development ecosystem.

CMake already contains a number of tools and features that facilitate development of the target software package. For example, CMake's `find_package` module simplifies dependency management, and the `CTest` package eases the process of testing the resulting software. Additional CMake tools can be created by writing CMake modules. While the CMake language is flexible and relatively simple, it is not without its pitfalls. Unfortunately, tools to facilitate the development of CMake modules are relatively sparse. Here we introduce CMinx, a tool for generating API documentation for CMake modules. API documentation is arguably one of the most basic elements of a software development ecosystem, and it is our hope that CMinx will serve as the foundation for a robust CMake development ecosystem.

Anecdotal evidence ([tjwrona1992, 2019](#)) indicates that Kitware, the developers of the CMake language, internally write their API documentation using reST and Sphinx. Following best practices, this reST documentation resides next to the described CMake code. Kitware has also written a Sphinx plugin that makes it easy to extract the API documentation as part of a normal Sphinx workflow. This Sphinx plugin is distributed with the source code for the CMake interpreter and is also available in a GitHub repository mirror ([Kitware, 2022](#)). For completeness, we note that similar Sphinx plugins ([Koch, 2020](#); [Lorenz, 2013](#)) have been independently developed but appear to now be abandoned.

To our knowledge, all of the aforementioned Sphinx plugins simply extract the reST API documentation verbatim. Notably, this means the developer is responsible for manually adding the function/macro signatures to the documentation, listing the function's parameters, and the overall formatting. For build system developers maintaining one or two CMake modules, these are admittedly pretty minor inconveniences; however, for organizations maintaining a substantial CMake code base (such as those for exascale programs), these "minor inconveniences" can impact productivity, particularly when ensuring consistency. CMinx differs from previous solutions primarily in three ways. First, CMinx understands the grammar of the CMake language, meaning CMinx can automatically generate some of the documentation by "reading" the source code. Second, CMinx generates static reST files; this decreases the number of stub files developers need to maintain and makes it easier for the resulting documentation to be used in workflows that do not rely on Sphinx. Finally, CMinx has a CMake API to integrate more easily into existing CMake workflows.

## Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

The authors would also like to acknowledge GitHub users dschiller, ni-dschiller, ni-fgenois, peanutfun, robertodr, and zachcran for suggestions, discussions, bug reports, and bug fixes.

## References

- Bartlett, R., Demeshko, I., Gamblin, T., Hammond, G., Heroux, M., Johnson, J., Klinvex, A., Li, X., McInnes, L. C., Moulton, J. D., Osei-Kuffuor, D., Sarich, J., Smith, B., Willenbring, J., & Yang, U. M. (2017). *xSDK foundations: Toward an extreme-scale scientific software development kit*. arXiv. <https://doi.org/10.48550/arXiv.1702.08425>
- Heesch, D. van. (2022). Doxygen. In *GitHub Repository*. GitHub. <https://github.com/doxygen/doxygen>

- Kitware. (2022). *Sphinx domain for modern CMake*. GitHub Repository. <https://github.com/scikit-build/moderncmakedomain>
- Koch, M. (2020). *Sphinx-cmake\_domain*. In *GitHub Repository*. GitHub. [https://github.com/MarcoKoch/sphinx-cmake\\_domain](https://github.com/MarcoKoch/sphinx-cmake_domain)
- Lorenz, K.-U. (2013). *GNU make domain*. In *GitHub Repository*. GitHub. <https://github.com/sphinx-contrib/cmakedomain>
- Pandoc: A universal document converter*. (2022). <https://pandoc.org/>
- reStructuredText: Markup syntax and parser component of docutils*. (2022). <https://docutils.sourceforge.io/rst.html>
- Snir, M., Wisniewski, R. W., Abraham, J. A., Adve, S. V., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., Chien, A. A., Coteus, P., DeBardleben, N. A., Diniz, P. C., Engelmann, C., Erez, M., Fazzari, S., Geist, A., Gupta, R., ... Hensbergen, E. V. (2014). Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28, 129–173. <https://doi.org/10.1177/1094342014522573>
- Sphinx: Python documentation generator*. (2022). <https://www.sphinx-doc.org/en/master/>
- tjwrona1992. (2019). What is the proper way to document a cmake module. In *Stack Overflow Forum*. StackOverflow. <https://stackoverflow.com/a/54671996>
- YAML: YAML ain't markup language*. (2022). <https://yaml.org>