The Journal of Open Source Software

# faust2clap: Generating CLAP Plug-ins from Faust DSP Code

**Facundo Franchino** [1], **Stéphane Letz** [2], **and Jatin Chowdhury** [3]

**1** School of Physics, Engineering and Technology, University of York, UK **2** GRAME-CNCM, Lyon, France **3** Department of Electrical Engineering and Computer Science, MIT, USA

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:** Open Journals

**Reviewers:**

- @openjournals

**Submitted:** 01 January 1970
**Published:** unpublished

## Summary

`faust2clap` is a tool that implements a direct compilation route from Faust DSP source code to plug-ins in the CLever Audio Plugin (CLAP) format. The system supports two modes of operation: static compilation, which produces optimised standalone plug-ins, and dynamic hot-reloading, which permits interactive development within the host environment. In the dynamic mode, DSP code may be modified and recompiled while the host continues to run, avoiding the repeated build–reload cycle that typically interrupts audio development workflows.

The implementation also provides automatic parameter discovery, MIDI support, polyphonic voice allocation, and state persistence. These features make the tool suitable both for researchers working with experimental DSP algorithms and for developers interested in rapid prototyping of audio effects and synthesisers within a modern, open plug-in standard.

## Statement of Need

The audio plug-in development ecosystem has traditionally required developers to navigate complex APIs and manage significant boilerplate code when creating plug-ins for digital audio workstations (DAWs). While Faust provides an elegant functional programming language for DSP algorithm design (Letz et al., 2018), and CLAP offers a modern, open-source plug-in standard (Free Audio Community, 2022), there has been no direct path between these two technologies.

`faust2clap` addresses this gap by providing researchers, audio developers, and educators with a streamlined workflow for plugin development. The tool is particularly valuable for:

- **Researchers** who wish to test novel DSP algorithms in DAWs without manually implementing plugin infrastructure
- **Educators** teaching DSP concepts who can now demonstrate algorithm behaviour directly within professional audio software
- **Audio developers** prototyping effects and synthesisers who benefit from Faust's compact DSP language and immediate plug-in generation

The hot-reload capability is especially significant for iterative development, allowing real-time experimentation with DSP parameters and algorithms while maintaining audio continuity in the host application. This feature eliminates the traditional compile-reload-test cycle that has historically slowed audio plugin development, enabling a more interactive and exploratory approach to DSP design.

## State of the Field

Several tools exist for generating audio plugins from Faust code, including `faust2vst`, `faust2au`, and `faust2lv2`, which target the VST, Audio Unit, and LV2 plugin standards respectively (Letz et al., 2024). However, these tools generate only statically compiled plugins and lack support for the modern CLAP standard, which offers advantages including better parameter automation, more flexible audio port configurations, and an open-source license without patent encumbrances.

The CLAP standard itself, developed by Bitwig and u-he (Free Audio Community, 2022), introduces improvements over earlier standards in plugin architecture design, but adoption has been limited by the lack of high-level development tools. While the clap-helpers library provides C++ utilities for CLAP development, it still requires substantial manual implementation work.

`faust2clap` extends the ecosystem through four contributions:

1. **First CLAP generator for Faust**: direct compilation of Faust DSP source code to CLAP plug-ins

2. **Dynamic hot-reload capability**: allows DSP code to be recompiled in real time without restarting the plug-in
3. **Automatic DSP type detection**: distinguishes between effects and instruments, configuring polyphonic behaviour accordingly
4. **GUI Support**: provides a Python interface for selecting DSP files and managing hot-reload operations

## Implementation and Architecture

The implementation comprises two distinct compilation pathways, each addressing specific deployment requirements. The static mode produces standalone plug-ins through Faust's C++ code generation, whilst the dynamic mode provides runtime DSP compilation using `libfaust`'s interpreter backend.

The static implementation centres on a specialised CLAP architecture file (`clap-arch.cpp`) that adapts Faust's generated DSP classes to the CLAP API. A custom `CLAPMapUI` class extends Faust's `MapUI` to capture parameter metadata during instantiation, storing ranges, default values, and zone pointers as parameters are declared. The Python orchestration script (`faust2clap.py`) manages the compilation process by invoking Faust with architecture-specific flags and coordinating the CMake build chain. The script performs automatic DSP classification by analysing input and output port counts, distinguishing between effect processors and synthesiser instruments to configure appropriate polyphonic behaviour.

The dynamic implementation employs `libfaust`'s interpreter virtual machine to compile DSP code at runtime. File modification monitoring through the efsw library triggers recompilation cycles whilst preserving audio processing continuity. The system maintains parameter states across DSP updates and manages instance switching to avoid audio artifacts during code transitions. Compiled bytecode executes on a stack-based virtual machine with separate integer and floating-point heaps for DSP state management.

Both modes share infrastructure for audio buffer handling and MIDI event processing. Format conversion between host audio representations (32-bit and 64-bit floating-point) and Faust's `FAUSTFLOAT` type occurs transparently. Polyphonic operation utilises Faust's `mydsp_poly` wrapper for voice allocation and MIDI event distribution. The build system integrates with the Faust ecosystem through standardised paths obtained via `faust --archdir`, maintaining compatibility across installation configurations.

## Usage and Adoption

`faust2clap` has recently been merged into the main Faust repository, where it is now distributed as part of the standard toolchain. Although still a new addition, its inclusion makes the functionality available to the wider Faust user community. The tool demonstrates how high-level DSP specifications in Faust can be compiled to a modern plug-in standard, and provides a reference point for future work on CLAP-based development tools.

## Acknowledgements

This work was supported by Google Summer of Code 2025. The author thanks Stéphane Letz (GRAME-CNCM) for project supervision and continuous support, Jatin Chowdhury (MIT) for CLAP expertise and mentorship, and the CLAP community for valuable feedback and testing. Special recognition goes to the Faust development team at GRAME for maintaining the Faust ecosystem that makes this work possible.

## References

Free Audio Community. (2022). *CLAP: CLever Audio Plug-in API*. https://github.com/free-audio/clap.

Letz, S., Orlarey, Y., & Fober, D. (2018). An overview of the faust developer ecosystem. *Proceedings of the 1st International Faust Conference (IFC-18)*.

Letz, S., Orlarey, Y., & Fober, D. (2024). *Faust architecture files*. https://faustdoc.grame.fr/manual/architectures/.