faust2clap: Generating CLAP Plug-ins from Faust DSP Code

By Facundo Franchino

Supervisors: Stéphane Letz & Jatin Chowdhury

This work was funded by the Google Summer of Code programme.
It was carried out from June until early September 2025.

Google
Summer of Code

Problem & Motivation

- Writing DSP code is already hard enough, and deploying it is no different
- Turning algorithms into plug-ins still requires lots of setup code
- Every DSP change means a full rebuild cycle

Traditional

What is Faust?

FAUST Functional Audio Stream

UNIVERSITY of York

- High-level DSP language for audio synthesis/effects
- Compiles to optimised C++ and other targets
- Rapid prototyping with built-in GUI & plugin generation
- Ideal for research, education, and real-time performance

What is CLAP?

CLAP

UNIVERSITY of York

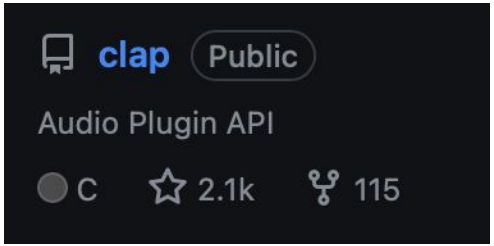
- Open-source modern plug-in standard (Bitwig, u-he)
- Stable ABI, per-voice control, better multithreading
- MIT-licensed and gaining support in major DAWs

Some of the already supported Digital Audio Workstations:

BITWIG STUDIO

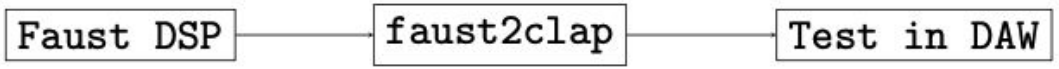
FL STUDIO

Studio One



Solution: faust2clap

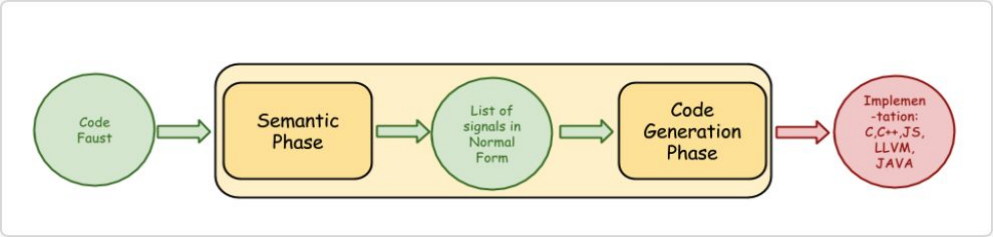
- **Two modes of compilation**:
  a. Static: turns any Faust .dsp file into a ready-to-use CLAP plugin
  b. Dynamic: a single plugin that hot-reloads any Faust code in real time, inside your DAW
- No boilerplate, or C++. Write DSP and get a plugin in seconds.
- Speeds up prototyping and development dramatically



faust2clap workflow

Under the Hood

- The Faust compiler turns .dsp files into an optimised C++ class
- faust2clap wraps this in a CLAP-compatible **architecture file**
- A Python build script handles plugin scaffolding, Makefile generation, and compilation



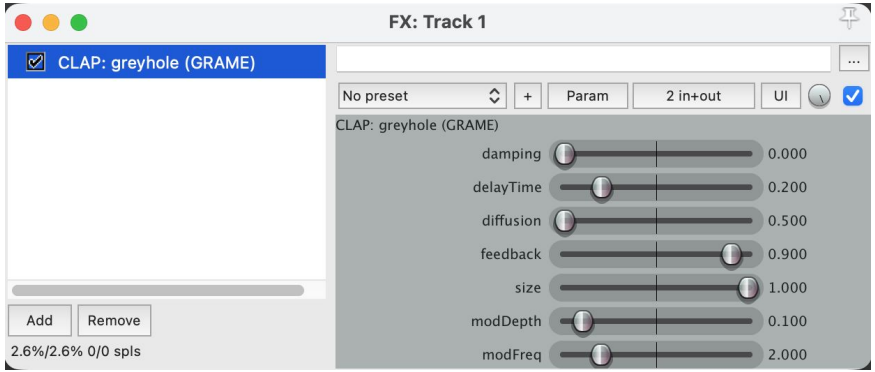
Dynamic compilation inside the DAW?

- Achieved via **libfaust**, the embeddable Faust compiler.
- libfaust supports multiple backends

Static Compilation

Build fully standalone CLAP plugins from Faust code.

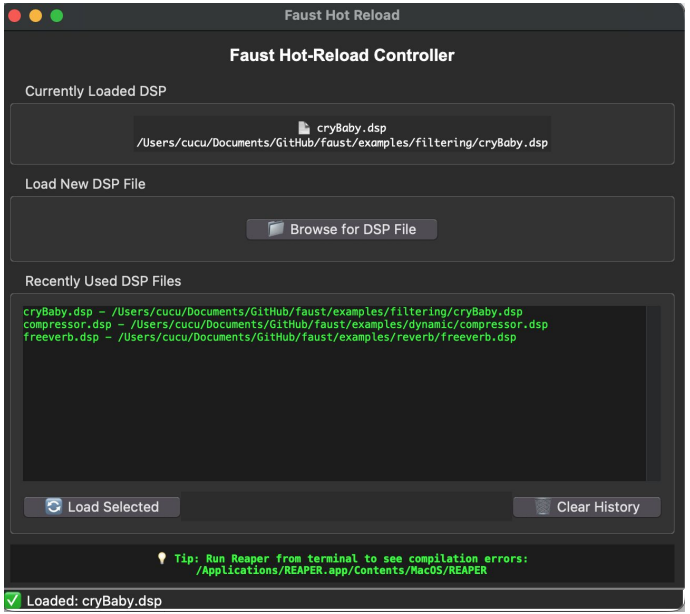
- Compiled to native code for performance.
- The result is a ready-to-use .clap binary with **zero manual coding**
- Built plugins are placed directly in your system's CLAP plugin directory
- Ready to use in your DAW



Compiling On-The-Fly: Hot-reloading

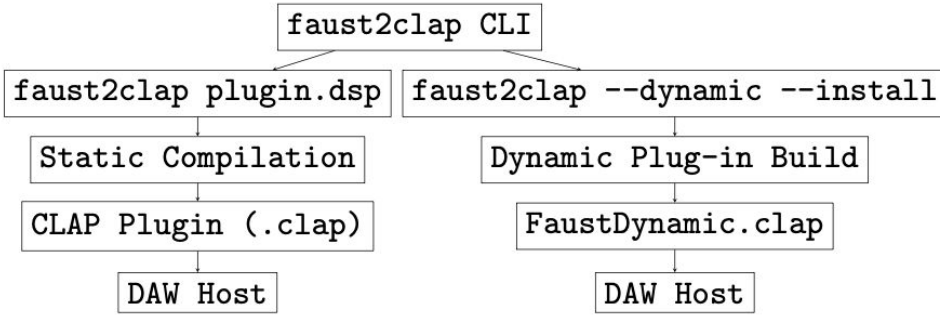
Edit your DSP code while the plug-in runs. No restart or reloading

- Uses an embedded interpreter for live editing.
- File watcher (EFSW) triggers → recompilation → live instance swap
- Maintains **audio continuity** across reloads
- Enables fast DSP iteration and live development inside the DAW



CLI & Workflow

- Global install script
- Plugins install directly to system plugin path



Who is this for?

Students and Educators

- Teach and learn DSP without writing plug-in code

Researchers

- Test algorithms in real-time inside a DAW

Developers

- Generate working CLAP plug-ins from DSP code
- Bridge prototyping and deployment in a single step

Live coders & musicians

- Build custom effects with minimal setup
- Modify DSP code live during playback

Future Work

Improvements:

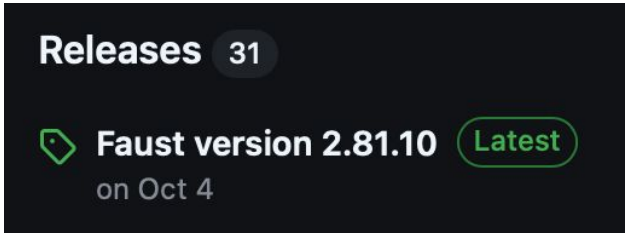
- Utilising LLVM instead of the Interpreter could make it 3 to 10 times faster.
- Needs a JUCE GUI implementation and a on-display preset management system which will end the need for an external .py script that runs the .dsp selector.
- Design choice of a 12 fixed-parameter system for the dynamic plugin places a limitation on dsp designs requiring more parameters (e.g a complex synthesiser).

Known bugs:

- All quad-channel and some analysis DSP's currently crash Reaper (specifically quadEcho,fourSourcesToOcto and vumeter.dsp).

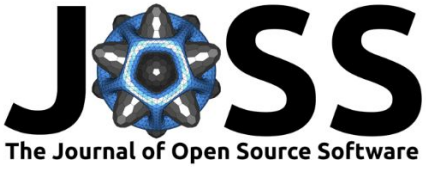
Conclusions

- Enables **instant** DSP prototyping
- Produces **production-ready** CLAP plugins
- Real contribution to **both** Faust & CLAP ecosystems
- Cross-platform support and integrated to Faust's latest release (2.81.10)

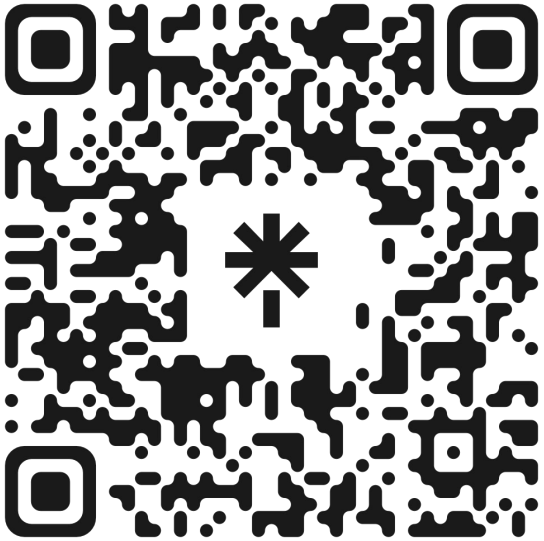


Acknowledgements

- Stéphane Letz and Jatin Chowdhury
- GRAME and Google Summer of Code for selecting me and funding this project
- CLAP community



References & Links



References:

[1] S. Letz, Y. Orlarey, and D. Fober, ‘An Overview of the FAUST Developer Ecosystem’, 2018.

[2]F. Franchino, S. Letz, and J. Chowdhury, ‘faust2clap: Generating CLAP Plug-ins from Faust DSP Code’, . *faust*, 2025.

Thank you for your attention!

Any questions?