Workshop

ADC 2023

Julian Storer Cesare Ferrari Lucas Thompson Harriet Drury

What we're going to cover:

- The basics: getting tooled-up
- Quick tour of Cmajor and glossary of terms
- Let's write a tremolo!
- Adding a custom GUI
- Turning it into a native VST/AU/CLAP plugin

Cmajor Tools

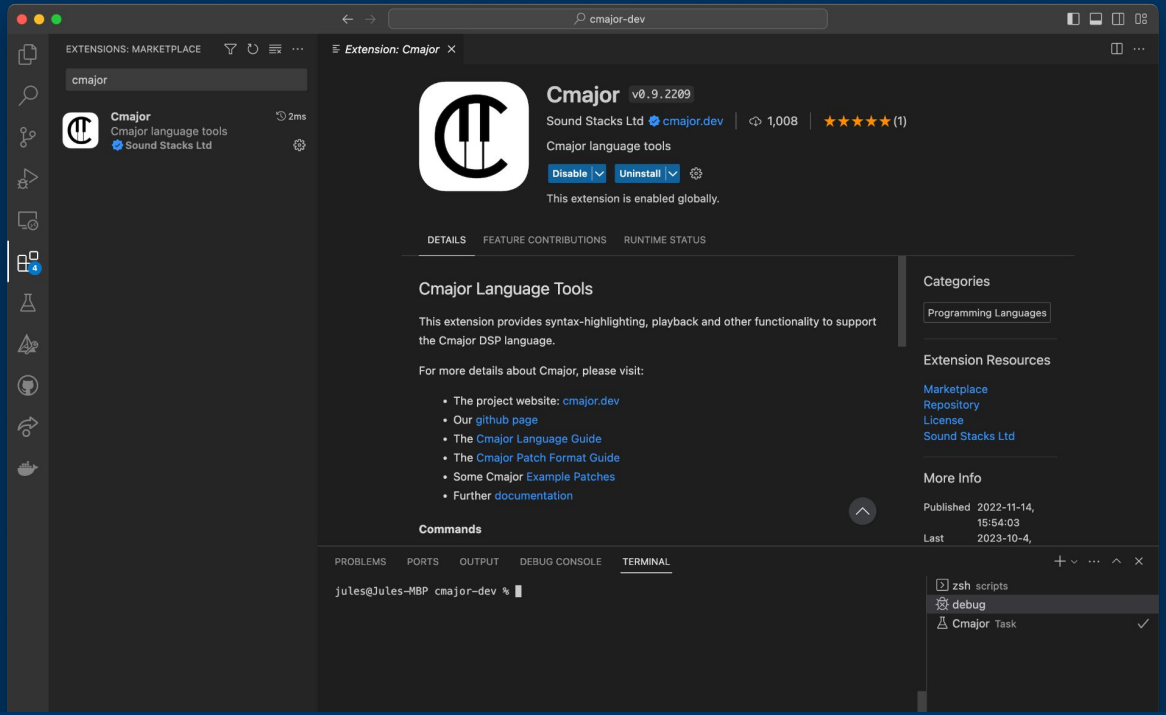
For this workshop you will need:

- Visual Studio Code
- Cmajor VSCode extension
- The workshop github - https://github.com/SoundStacks/ADC23_workshop
- For plugin builds, IDE of your choice (e.g. Xcode)

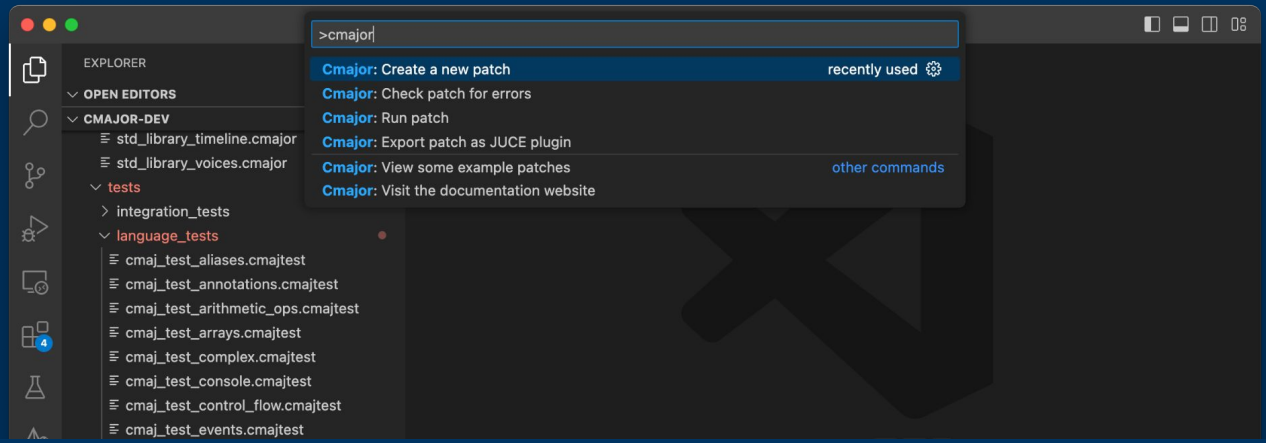
Optional stuff:

- Command-line tool
- Loader Plugin
- ML tools

Cmajor Tools



Cmajor Tools



Cmajor Glossary

- C-family language style
- JIT and live-coding
- Patch format
- Graphs and Processors
- Endpoints: streams, events and values
- GUIs using HTML/Javascript
- Export to C++/JUCE/CLAP/Javascript

Let’s write a tremolo!

We’re going to walk through the steps involved in creating a tremolo effect patch.

https://github.com/SoundStacks/ADC23_workshop

What is the Tremolo effect?

Applies amplitude modulation to a signal

- Rate affects modulation speed
- Depth affects modulation amount
- Waveform control morphs between triangle and square waveforms
- Depressing the pedal bypasses the effect

What DSP is required?

Two components are required:

- Amplitude modulation
- LFO

Amplitude modulation is trivial, the effort will be producing an LFO with suitable behaviour and control input.

We want the LFO to morph between the two waveforms, triangle and square, to give different types of modulation characteristics

Stage 1

Let's start by writing a triangle LFO which allow the rate to be varied in a suitable range (0.1Hz to 20Hz).

The standard library contains an LFO component, see https://cmajor.dev/docs/StandardLibrary#LFO

Steps:

- Create a new patch
- Add an input parameter to represent the rate
- Add an output stream containing the LFO's output level
- Add a graph node for the standard library LFO component
- Add connections to route the parameter to the LFO, and the LFO output to the output stream

Stage 2

Let’s extend the LFO to include a depth control.

The depth control on the GUI will be 0..100 to represent a % depth, whilst the LFO amplitude needs to be in the 0..1 range.

Steps:

- Add an event handler which will be triggered when this parameter changes, and will update the LFO.

Stage 3

Extend the LFO to include a square LFO, and add a parameter to morph between the two shapes. This control will range from 0 (triangle) to 100 (square).

Steps:

- Add a ‘shape’ parameter
- Add a second LFO producing a square wave
- Route the rate and depth to this new LFO
- Mix the two LFOs to the output using the shape parameter to morph between them
- Consider adding a filter to remove the sharp edges from the square oscillator

Stage 4

Add the Tremolo component, applying the modulation to an audio stream. Add a bypass control to enable/disable the Tremolo

Steps:

- Add a Tremolo graph, with an input and output stream, and bypass parameter
- Add the LFO as a node, and add the LFO parameters
- Add a function to calculate the required gain given the bypass and lfo values
- Add a connection to generate the tremolo effect

Testing Cmajor code

What we want to achieve:

- Validate DSP behaves as expected
- Reproducible tests which can be integrated into CI
- Support regression testing of patches
- Extensible testing framework to support custom tests

What is not supported:

- No support for GUI testing

Testing Cmajor code

https://cmajor.dev/docs/TestFileFormat

.cmajtest file format

```
## testProcessor()

processor P [[ main ]]
{
    output event int32 out;
    void main() { out <- 1; advance(); out <- -1; }
}

## expectError ("4:19: error: Endpoint types cannot be references")

processor P [[ main ]]
{
    output stream float32& out;
    void main() { out <- 1.0f; advance(); out <- -1.0f; }
}

## testFunction()

bool testVectorAdd()
{
    int<4> f = (1, 2, 3, 4);
    let g = f + f;

    return allTrue (g == int<4> (2, 4, 6, 8));
}
```

- Tests introduced by ‘##’
- Multiple tests can be included in one test file
- Test files can be run from within VSCode using ‘run’ (F5)
- Tests are javascript functions, see tests/cmaj_test_functions.js for implementations

Testing Cmajor code

Demonstration

- Running the example tests
- Adding an additional test
- Integrating with CI (command line option)

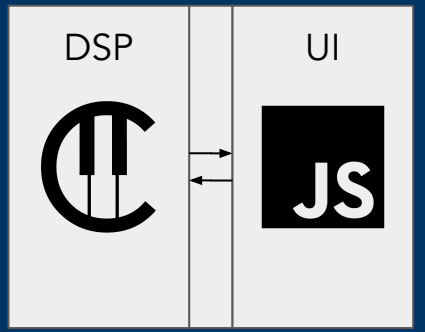
Adding a custom GUI

Let’s add a Patch view!
https://cmajor.dev/docs/PatchFormat#patch-guis

What is a Patch view?

A UI implemented with web technologies

- PatchConnection API encapsulates Engine <-> UI communication
  - JSON structures sent over a communication channel
- Same code runs in different environments
  - Plugin, vscode, Web Audio Worklet
- Hot reloading as with Cmajor



Building UIs with the PatchConnection

So, back to the editor! We'll focus on interacting with our API, and gain some intuition for web UIs along the way.

We'll roughly do the following:

- Add a view property to the manifest
- Create our view entry point
- Refresh our JS knowledge
- Experiment interactively to build a basic UI + interact with the connection
- Wire up an existing stompbox widget

Exporting the finished patch

- Using the patch loader plugin
- Exporting a JUCE plugin for VST/AU
- Exporting to CLAP
- Exporting to a webpage
- Embedding the Cmajor JIT engine