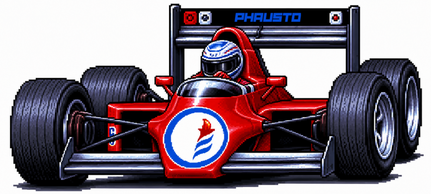
PHAUST02JUCE:

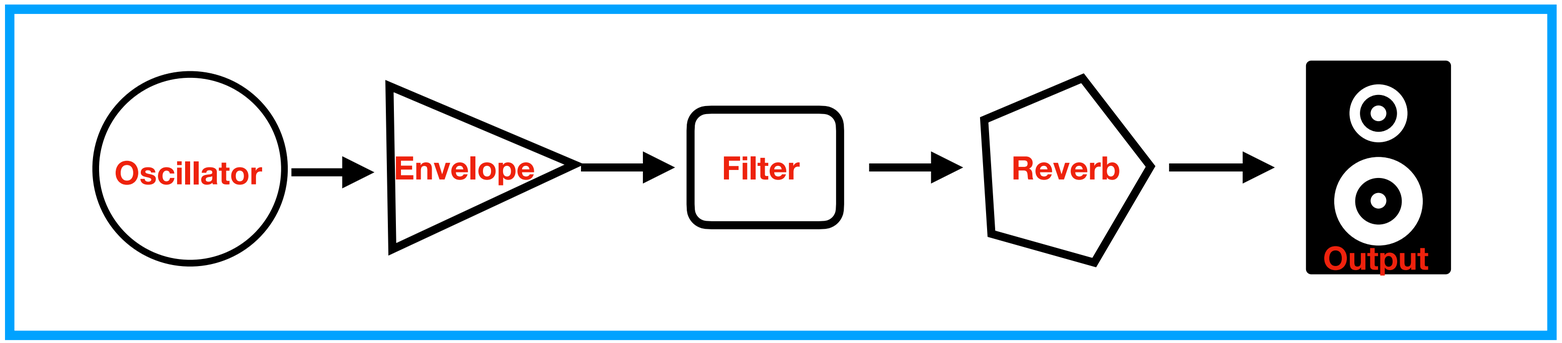
DESIGNING AND DEPLOYING AUDIO PLUGINS WITH PHAUSTO

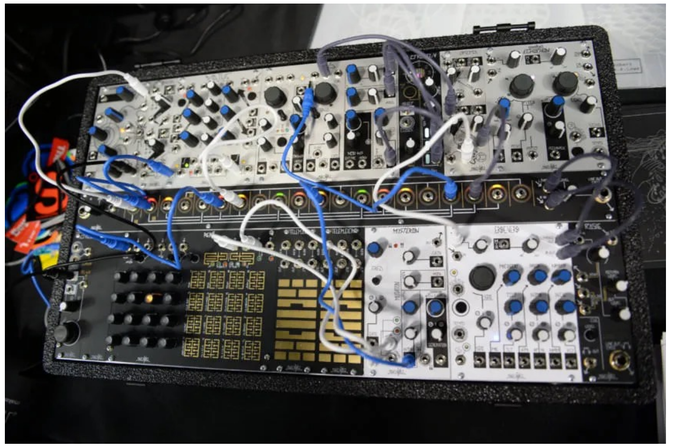
Domenico Cipriani, 2026

What is Phausto

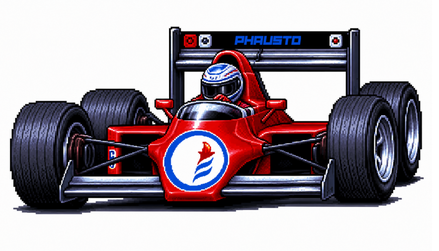


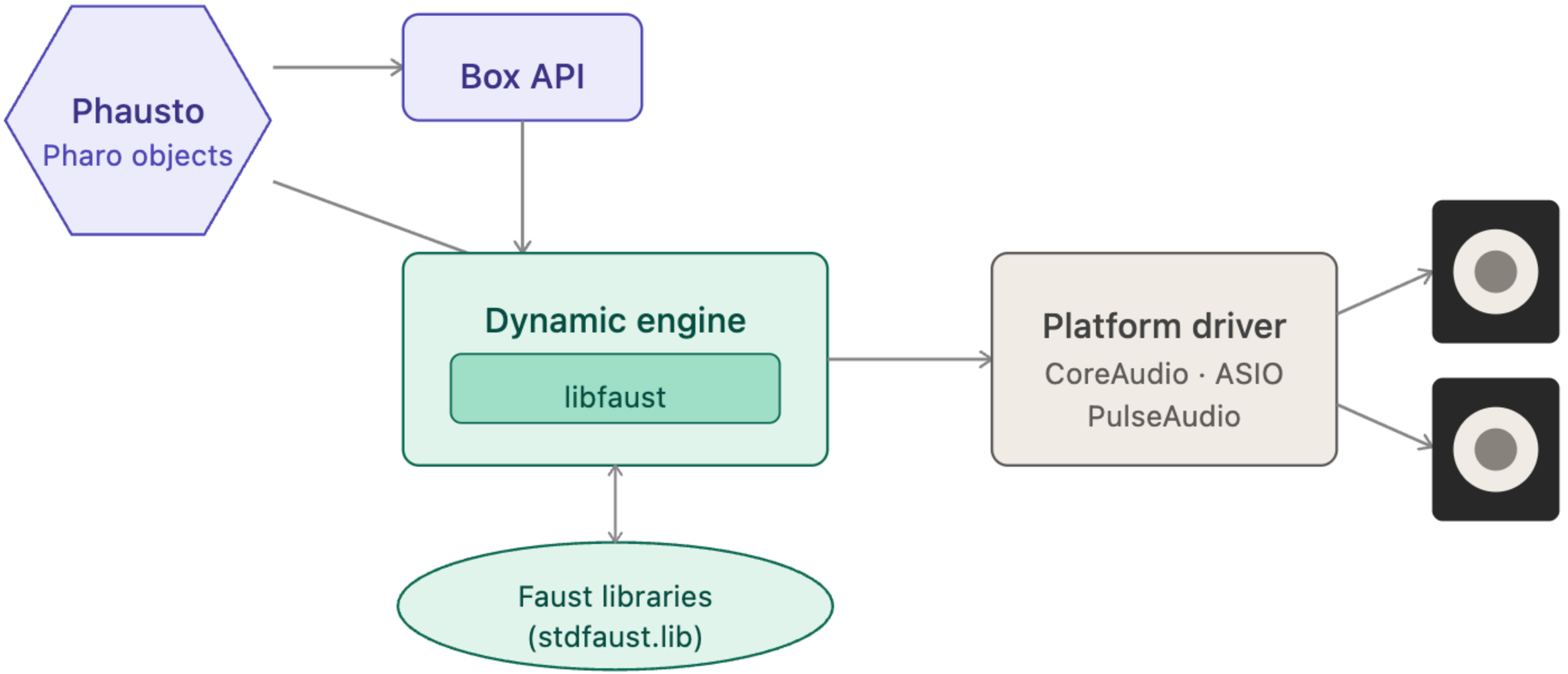
- Phausto is a library and API for sound generation and DSP programming within the Pharo IDE, powered by an embedded Faust compiler.
- Phausto has a modular-synth-inspired approach to designing synths and effects.
- Unit Generators are connected by setting their members value or using the ChucK operator => .
- A dynamic C library, tailored by Stephane Letz, computes the output of the synthesisers and effects created with Phausto, using the FAUST compiler.



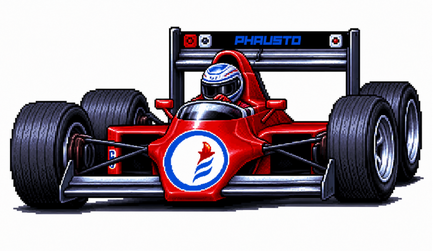


Inside Phausto



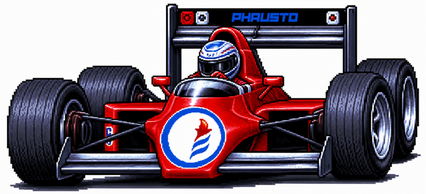


What is Smalltalk?



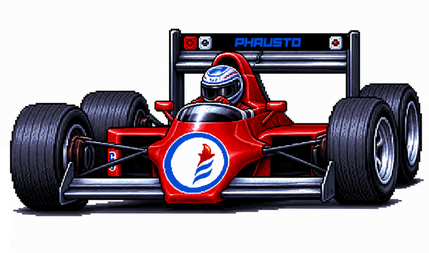
- A true Object-Oriented Programming language developed at the Learning Research Group at Xerox PARC in the 1970s by Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaeheler.
- Designed for educational use following principles of Constructionism.
- Deeply influenced by Simula, developed by Ole-Johan Dahl and Kristen Nygaard in the 1960s at the Norwegian Computing Center in Oslo.
- Not just a language, also an IDE users can inspect and modify.
- Programs written in Smalltalk are compiled into byte code and interpreted by a virtual machine.
- Smalltalk has influenced Objective C, Ruby, SuperCollider.

What is Pharo?



- A true object-oriented, dynamically typed, and reflective language with a platform independent IDE.
- Pharo was born as a fork of Squeak Smalltalk, led by Stéphane Ducasse and Marcus Denker at INRIA Lille.
- It is currently developed by INRIA, the Pharo Consortium, and a vibrant worldwide community
- Its syntax fits in a postcard.
- Integrated Git support and a framework for SUnit Tests.

Phao syntax fits on a postcard



Pharo

```
exampleWithNumber: x

<syntaxOn: #postcard>
"A ""complete"" Pharo syntax"
| y |
true & false not & (nil isNil)
 ifFalse: [ self perform: #add: with: x ].
y := thisContext stack size + super size.
byteArray := #[2 2r100 8r20 16rFF].
{ -42 . #($a #a #'I''m' 'a' 1.0 1.23e2 3.14s2 1) }
 do: [ :each |
  | var |
  var := Transcript
   show: each class name;
   show: each printString ].

^ x < y
```

https://www.pharo.org

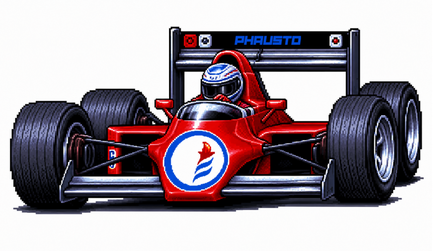
PLACE STAMP HERE

Rule 1: Everything is an Object.
Rule 2: Every Class has a superclass.
Rule 4: Everything happens by sending messages.
Rule 5: Method lookup follows inheritance chain.
Rule 6 : Classe are Objects too and they follow the same rules.

Precedence rules:

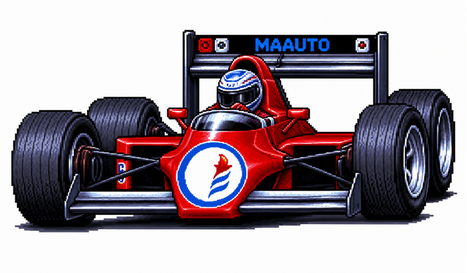
1. Unary message (3 factorial)
2. Binary messages (3 + 5)
3. Keyword messages (Transcript show: ‘Hello’).
4. Multiple messages with the same precedence are evaluated **from left to right**.

Pharo's shortcuts



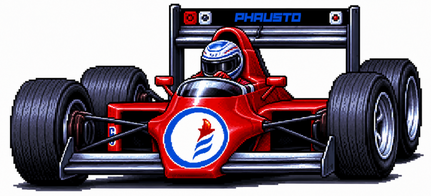
CMD/CTRL + OP	Open Playground
CMD/CTRL + D	Evaluate (Do)
CMD/CTRL + P	Print Object
CMD/CTRL + I	Inspect Object
CMD/CTRL + B	Browse Object
CMD/CTRL + M	List Implementors

What is JUCE?



- Open Source C++ Application Development Framework.
- Deliver software on multiple platforms.
- Integrate into different build systems (CMake, VSCode, Xcode...)
- Deploy to different devices
- Target multiple plug-in formats (VST2, AU, AAX...).
- Modules for GUI design, MIDI and OSC integration, Cryptography, and more

Phausto API

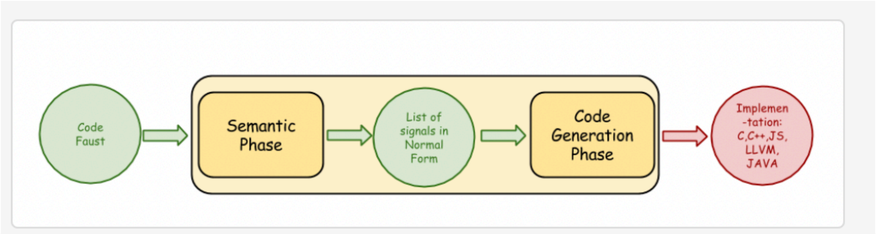


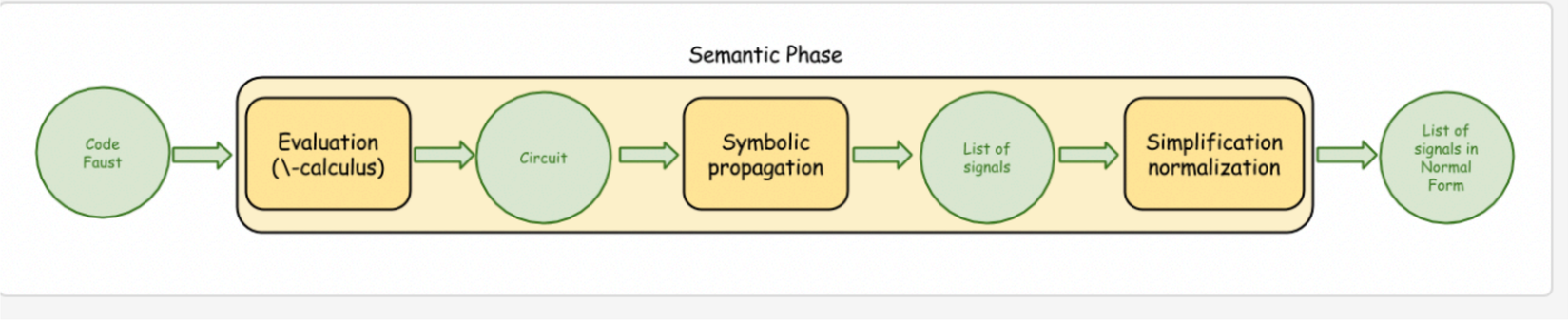
- Phausto API is designed to be straightforward.

- The subdivision of Faust's standard library functions into Phausto Unit Generators subclasses is deeply inspired by the ChucK programming language, which in turn takes this concept from MUSIC-N style programming music language.

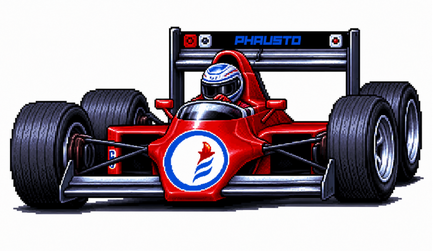
- UnitGenerators in Phausto are implemented using the Faust Box API.

- The FAUST C box API allows for the programmatic creation of a box expression, which is used to create a DSP object. This stage is a new intermediate public entry point created in the Semantic Phase of FAUST's compilation chain.





Ported & exportable



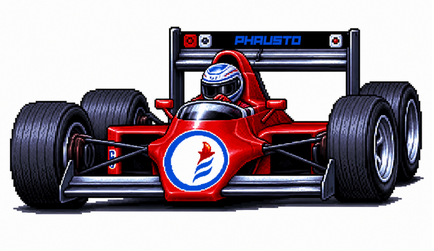
80%

of the FAUST libraries ported into Phausto

Phausto can also build audio plugins:

- **JUCE exporter** Generate plugins through the JUCE framework
- **Cmajor exporter** Target the Cmajor audio platfor
- *Write DSP in Pharo → ship a real plugin.*

From Phausto to JUCE



Once a valid DSP is created in Phausto we send the message exportWithName: aString type: aPluginType

Playground

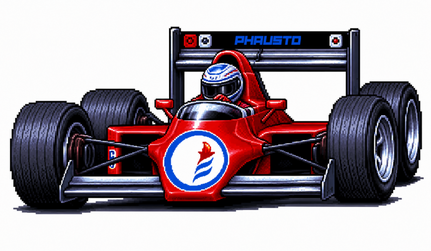
Do it all | Publish | Bindings | Versions | Pages

```
1  pitchShiftL := (PhHSlider new label: 'ShiftL' values: #(0 -12 12 0.1)) * (PhButton new label: 'leftEye').
2  pitchShiftR := (PhHSlider new label: 'ShiftR' values: #(0 -12 12 0.1) )* (PhButton new label: 'rightEye').
3
4  filter := OberheimBPF new label: 'Oberheim'.
5  transposerL := (Wire new => filter )=> (PhTranspose new shift: pitchShiftL).
6  transposerR := (Wire new => filter ) => (PhTranspose new shift: pitchShiftR).
7
8  reverb := (GreyHole  new label: 'Reverb' ; inputL: transposerL ; inputR: transposerR) .
9
10 dsp := reverb asDsp
11 dsp init.
12 dsp start.
13 dsp stop.
14 dsp export2JuceWithName: 'SquidVerb' type: 'Effect'.
15
16 dsp displayUI.
```

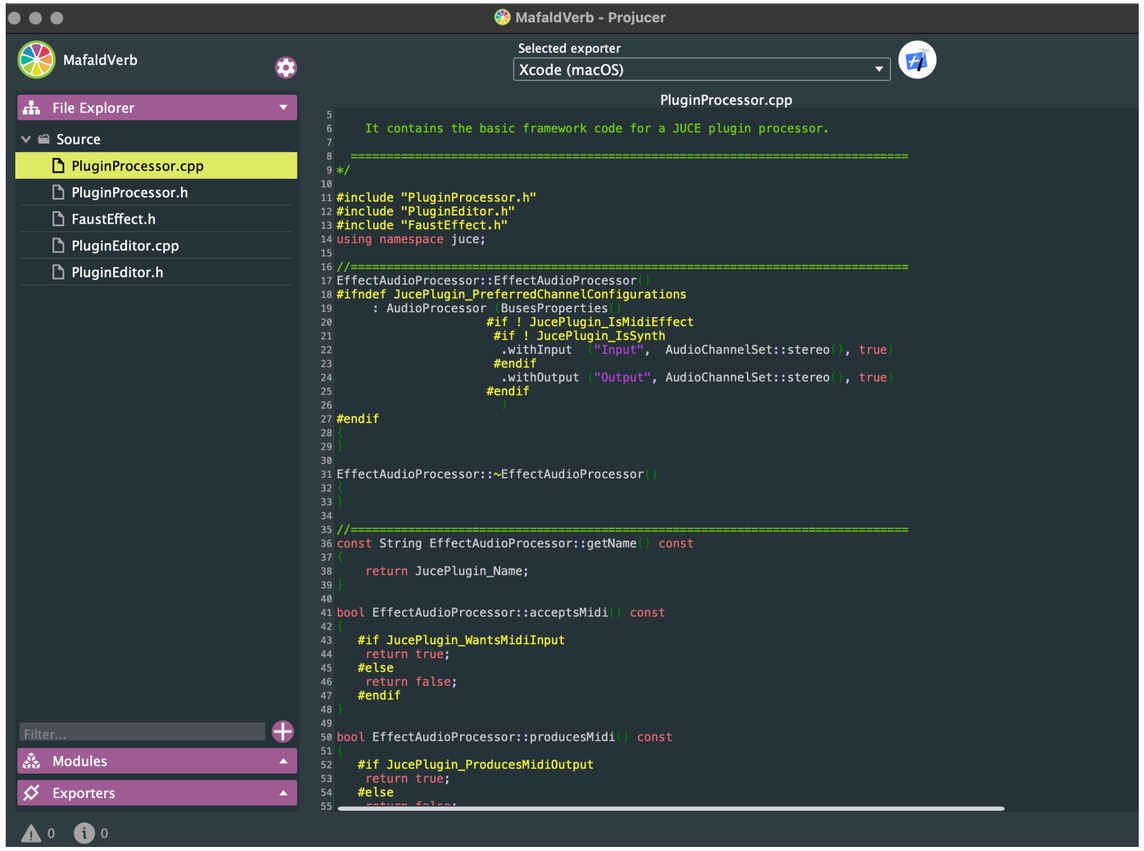
A folder is created inside Documents/faust2Juce/

PluginProcessor.h	Today at 16:36	3 KB	C Head...Source
PluginEditor.cpp	Today at 16:32	1 KB	C++ Source
PluginEditor.h	Today at 16:30	1 KB	C Head...Source
FaustEffect.h	Today at 11:03	76 KB	C Head...Source
PluginProcessor.cpp	Today at 11:03	7 KB	C++ Source

From Phausto to JUCE

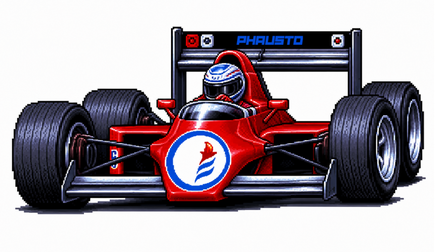


A .jucer file is generated



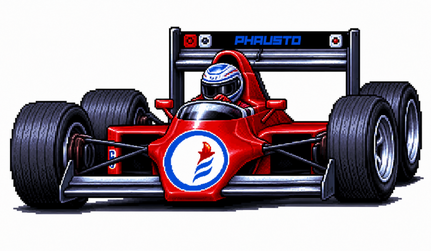
And also a CMakeLists.txt file!

```
# CMakeLists.txt generated by Phausto for a Faust/JUCE plugin.
# Configure + build:  cmake -B build -DCMAKE_BUILD_TYPE=Release && cmake --build build
# Use a local JUCE instead of fetching:  cmake -B build -DJUCE_PATH=/path/to/JUCE
# (Fetched JUCE is pinned to tag 8.0.4 below; bump GIT_TAG to your target version.)

cmake_minimum_required(VERSION 3.22)

project(apvtsTEST VERSION 1.0.0)

if(DEFINED JUCE_PATH)
    add_subdirectory(${JUCE_PATH} JUCE)
else()
    include(FetchContent)
    FetchContent_Declare(JUCE
        GIT_REPOSITORY https://github.com/juce-framework/JUCE.git
        GIT_TAG 8.0.4)
    FetchContent_MakeAvailable(JUCE)
endif()

juce_add_plugin(apvtsTEST
    COMPANY_NAME "Phausto"
    BUNDLE_ID com.phausto.faustplugin
    IS_SYNTH FALSE
    NEEDS_MIDI_INPUT FALSE
    NEEDS_MIDI_OUTPUT FALSE
    IS_MIDI_EFFECT FALSE
    EDITOR_WANTS_KEYBOARD_FOCUS FALSE
    COPY_PLUGIN_AFTER_BUILD TRUE
    PLUGIN_MANUFACTURER_CODE Phau
    PLUGIN_CODE Fplu
    FORMATS AU VST3 Standalone
    VST3_CATEGORIES Fx
    AU_MAIN_TYPE kAudioUnitType_Effect
    PRODUCT_NAME "apvtsTEST")

# AAX (the .jucer had buildAAX=1): add AAX to FORMATS above, then set the SDK path:
#   juce_set_aax_sdk_path(/path/to/aax-sdk)

juce_generate_juce_header(apvtsTEST)

target_sources(apvtsTEST
    PRIVATE
        Source/PluginProcessor.cpp
        Source/PluginEditor.cpp)

target_compile_features(apvtsTEST PUBLIC cxx_std_17)

target_compile_definitions(apvtsTEST
    PUBLIC
        JUCE_WEB_BROWSER=0
        JUCE_USE_CURL=0
        JUCE_VST3_CAN_REPLACE_VST2=0)
```



All the magic happens inside FaustEffect.h/FaustSynth.h - that are generated by the FAUST compiler with a minimal architecture file

```
669  class mydsp : public dsp {
1348     virtual void buildUserInterface(UI* ui_interface) {
1349         ui_interface->openVerticalBox("MyApp");
1350         ui_interface->addHorizontalSlider("OberheimNormFreq", &fHslider3, FAUSTFLOAT(1.0f), FAUSTFLOAT(0.0f), FAUSTFLOAT(1.0f),
                 FAUSTFLOAT(0.001f));
1351         ui_interface->addHorizontalSlider("OberheimQ", &fHslider4, FAUSTFLOAT(1.0f), FAUSTFLOAT(1.0f), FAUSTFLOAT(25.0f),
                 FAUSTFLOAT(0.707f));
1352         ui_interface->addHorizontalSlider("ReverbDamp", &fHslider0, FAUSTFLOAT(0.3f), FAUSTFLOAT(0.0f), FAUSTFLOAT(1.0f),
                 FAUSTFLOAT(0.001f));
1353         ui_interface->addHorizontalSlider("ReverbDt", &fHslider9, FAUSTFLOAT(1.0f), FAUSTFLOAT(0.1f), FAUSTFLOAT(6e+01f), FAUSTFLOAT(0.1f));
1354         ui_interface->addHorizontalSlider("ReverbEarlyDiff", &fHslider2, FAUSTFLOAT(0.3f), FAUSTFLOAT(0.0f), FAUSTFLOAT(1.0f),
                 FAUSTFLOAT(0.001f));
1355         ui_interface->addHorizontalSlider("ReverbFeedback", &fHslider6, FAUSTFLOAT(0.3f), FAUSTFLOAT(0.0f), FAUSTFLOAT(1.0f),
                 FAUSTFLOAT(0.001f));
1356         ui_interface->addHorizontalSlider("ReverbModDepth", &fHslider7, FAUSTFLOAT(0.3f), FAUSTFLOAT(0.0f), FAUSTFLOAT(1.0f),
                 FAUSTFLOAT(0.001f));
1357         ui_interface->addHorizontalSlider("ReverbModFreq", &fHslider8, FAUSTFLOAT(0.1f), FAUSTFLOAT(0.1f), FAUSTFLOAT(1e+01f),
                 FAUSTFLOAT(0.001f));
1358         ui_interface->addHorizontalSlider("ReverbSize", &fHslider1, FAUSTFLOAT(2.0f), FAUSTFLOAT(0.5f), FAUSTFLOAT(3.0f),
                 FAUSTFLOAT(0.001f));
1359         ui_interface->addHorizontalSlider("ShiftL", &fHslider10, FAUSTFLOAT(0.0f), FAUSTFLOAT(-12.0f), FAUSTFLOAT(12.0f), FAUSTFLOAT(0.1f));
1360         ui_interface->addHorizontalSlider("ShiftR", &fHslider5, FAUSTFLOAT(0.0f), FAUSTFLOAT(-12.0f), FAUSTFLOAT(12.0f), FAUSTFLOAT(0.1f));
1361         ui_interface->addButton("leftEye", &fButton1);
1362         ui_interface->addButton("rightEye", &fButton0);
1363         ui_interface->closeBox();
1364     }
1365
1366     virtual void compute(int count, FAUSTFLOAT** RESTRICT inputs, FAUSTFLOAT** RESTRICT outputs) {
1367         FAUSTFLOAT* input0 = inputs[0];
1368         FAUSTFLOAT* input1 = inputs[1];
1369         FAUSTFLOAT* output0 = outputs[0];
1370         FAUSTFLOAT* output1 = outputs[1];
1371         float fSlow0 = float(fHslider0);
1372         float fSlow1 = float(fHslider1);
1373         int iSlow2 = itbl0mydspSIG0[int(48.0f * fSlow1)];
1374         float fSlow3 = 0.0001f * float(iSlow2);
```



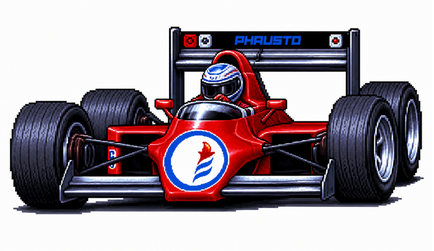
The Phausto exporter populates the AudioProcessor with basic implementations and definitions of a set of functions that act as an entry point to change the DSP parameters

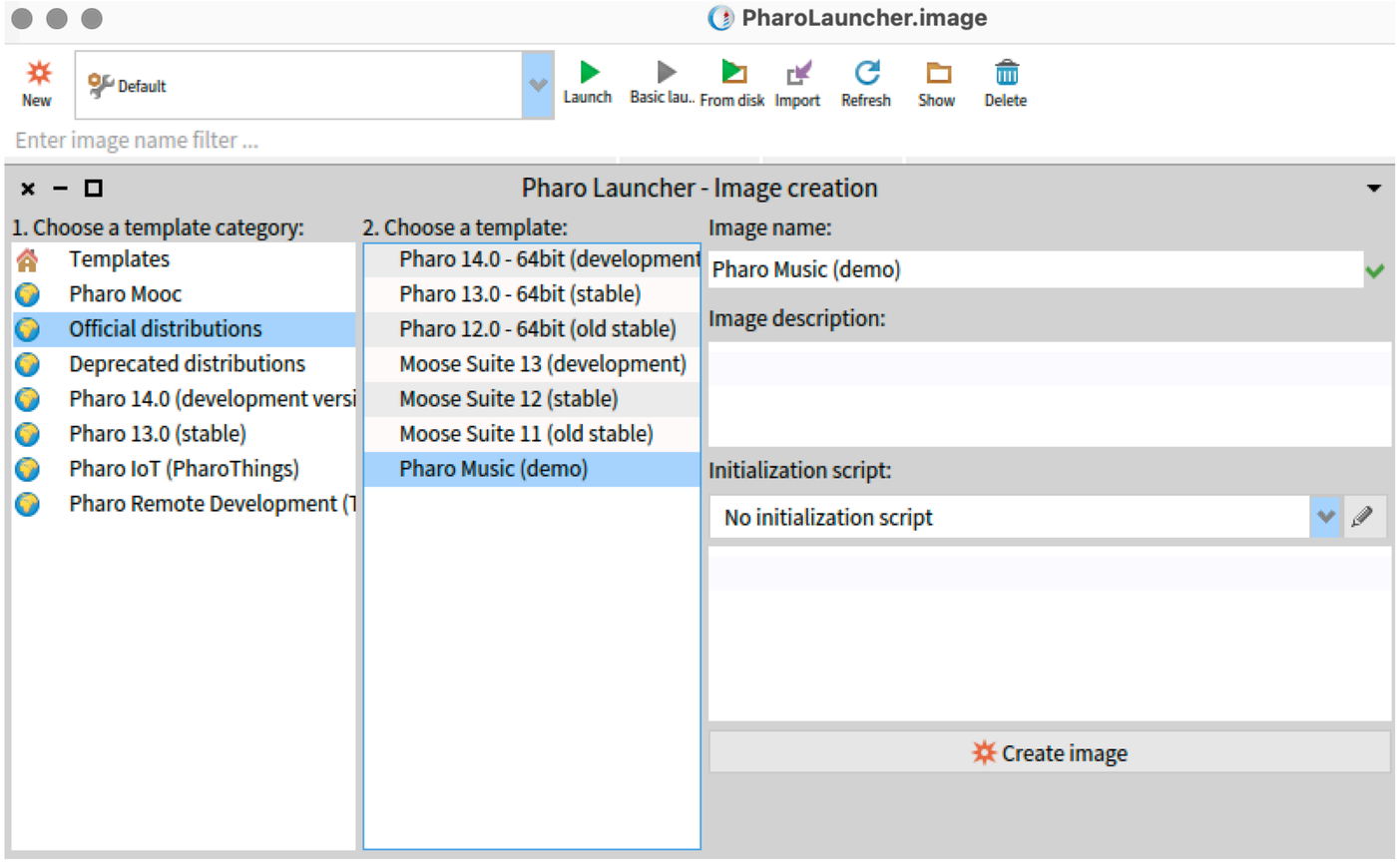
```
60      void getStateInformation (MemoryBlock& destData) override;
61      void setStateInformation (const void* data, int sizeInBytes) override;
62      void setOberheimNormFreq(float oberheimNormFreq);
63      void setOberheimQ(float oberheimQ);
64      void setReverbDamp(float reverbDamp);
65      void setReverbDt(float reverbDt);
66      void setReverbEarlyDiff(float reverbEarlyDiff);
67      void setReverbFeedback(float reverbFeedback);
68      void setReverbModDepth(float reverbModDepth);
69      void setReverbModFreq(float reverbModFreq);
70      void setReverbSize(float reverbSize);
71      void setShiftL(float shiftL);
72      void setShiftR(float shiftR);
```



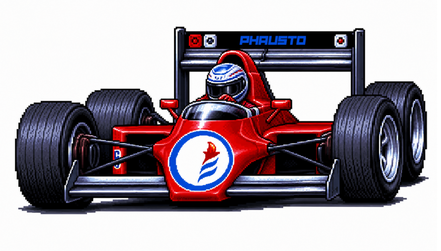
```
209 void EffectAudioProcessor::setOberheimNormFreq(float oberheimNormFreq) { fUI->setParamValue("OberheimNormFreq",oberheimNormFreq); }
210 void EffectAudioProcessor::setOberheimQ(float oberheimQ) { fUI->setParamValue("OberheimQ",oberheimQ); }
211 void EffectAudioProcessor::setReverbDamp(float reverbDamp) { fUI->setParamValue("ReverbDamp",reverbDamp); }
212 void EffectAudioProcessor::setReverbDt(float reverbDt) { fUI->setParamValue("ReverbDt",reverbDt); }
213 void EffectAudioProcessor::setReverbEarlyDiff(float reverbEarlyDiff) { fUI->setParamValue("ReverbEarlyDiff",reverbEarlyDiff); }
214 void EffectAudioProcessor::setReverbFeedback(float reverbFeedback) { fUI->setParamValue("ReverbFeedback",reverbFeedback); }
215 void EffectAudioProcessor::setReverbModDepth(float reverbModDepth) { fUI->setParamValue("ReverbModDepth",reverbModDepth); }
216 void EffectAudioProcessor::setReverbModFreq(float reverbModFreq) { fUI->setParamValue("ReverbModFreq",reverbModFreq); }
217 void EffectAudioProcessor::setReverbSize(float reverbSize) { fUI->setParamValue("ReverbSize",reverbSize); }
218 void EffectAudioProcessor::setShiftL(float shiftL) { fUI->setParamValue("ShiftL",shiftL); }
219 void EffectAudioProcessor::setShiftR(float shiftR) { fUI->setParamValue("ShiftR",shiftR); }
```

Download the latest Pharo Launcher





Or the PharoMusic image (latest)



And the launcher from:
https://pharo.org/download