MUSIC 420B Project
Implementing “FAUST to Audio Unit”

Reza Payami

1. INTRODUCTION

A Faust program describes a signal processor, a pure computation that maps input signals to output signals. It says nothing about audio drivers or GUI toolkits. This missing information is provided by architecture files. An architecture file describes how to relate a Faust program to the external world, in particular the audio drivers and the user interface to be used. This approach allows a single Faust program to be easily deployed to a large variety of audio standards. The goal of this project is to implement the “Audio Unit” architecture file in order to automatically generate Audio Units based on the given Faust code.

2. ROADMAP

Accomplished Activities

(in general for simplicity FaustAU and FaustAUEffect are used interchangeably)

- Read FAUST manual, tutorials and examples, and the architecture file pattern
- Studied VST SDK, the existing VST architecture file, and the generated VST source code
- Studied Audio Unit SDK, some existing examples, and implemented some Audio Units
- Developed the initial Xcode project containing the architecture file and other required classes
- Specified and implemented the initial generalization to implement the architecture file as a wrapper Audio Unit to call FAUST classes
- Used different “mydsp.cpp” files compiled by FAUST, in the Xcode project, to generate and test some initial Audio Units
- Adding the multi-channel support
- Measuring the latency of the generated Audio Units
- Implementing the required commands and bash scripts to have **faust2au** command and the build process to automatically generate the Audio Units
- Implementing **faust2ausynth** which polyphony support by implementing all the required classes and performing the similar activities as described in the above steps
- Generating Audio Units for the existing FAUST programs and testing the results

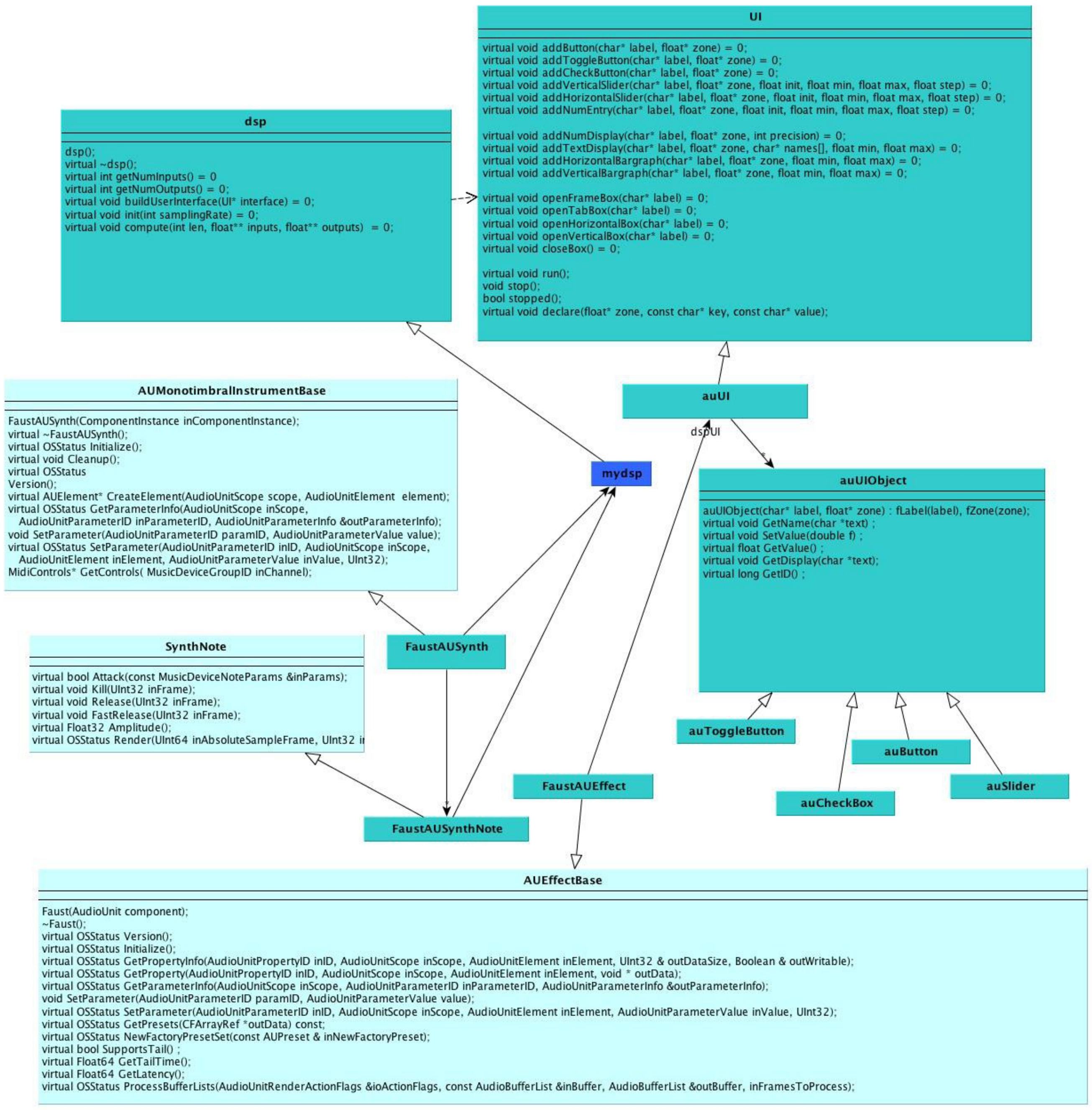
3. GITHUB REPOSITORY

All the files have been committed under the following GitHub address:

https://github.com/rpayami/faust2au

4. CLASS DIAGRAM

The following figure shows the main classes used for implementing faust2au and faust2ausynth. For simplicity, the overriden moethods are only shown in the base classes. AUMonotimbralInstrumentBase and SynthNote class are the based classed required for implementing an AU synth. AUEffect base is the corresponding class for writing an AU effect.'faust -a' command generates 'myDSP' class which is included in the source file of FaustAUEffect or FaustAUSynth. These two classes are the main classes in the realted architecture files, au.cpp and ausynth.cpp. In general all the classes with the same color as these two classes are implemented inside the architecture files.



```
UI

virtual void addButton(char* label, float* zone) = 0;
virtual void addToggleButton(char* label, float* zone) = 0;
virtual void addCheckButton(char* label, float* zone) = 0;
virtual void addVerticalSlider(char* label, float* zone, float init, float min, float max, float step) = 0;
virtual void addHorizontalSlider(char* label, float* zone, float init, float min, float max, float step) = 0;
virtual void addNumEntry(char* label, float* zone, float init, float min, float max, float step) = 0;

virtual void addNumDisplay(char* label, float* zone, int precision) = 0;
virtual void addTextDisplay(char* label, float* zone, char* names[], float min, float max) = 0;
virtual void addHorizontalBargraph(char* label, float* zone, float min, float max) = 0;
virtual void addVerticalBargraph(char* label, float* zone, float min, float max) = 0;

virtual void openFrameBox(char* label) = 0;
virtual void openTabBox(char* label) = 0;
virtual void openHorizontalBox(char* label) = 0;
virtual void openVerticalBox(char* label) = 0;
virtual void closeBox() = 0;

virtual void run();
void stop();
bool stopped();
virtual void declare(float* zone, const char* key, const char* value);
```

```
dsp

dsp();
virtual ~dsp();
virtual int getNumInputs() = 0
virtual int getNumOutputs() = 0;
virtual void buildUserInterface(UI* interface) = 0;
virtual void init(int samplingRate) = 0;
virtual void compute(int len, float** inputs, float** outputs)  = 0;
```

```
AUMonotimbralInstrumentBase

FaustAUSynth(ComponentInstance inComponentInstance);
virtual ~FaustAUSynth();
virtual OSStatus Initialize();
virtual void Cleanup();
virtual OSStatus
Version();
virtual AUElement* CreateElement(AudioUnitScope scope, AudioUnitElement  element);
virtual OSStatus GetParameterInfo(AudioUnitScope inScope,
   AudioUnitParameterID inParameterID, AudioUnitParameterInfo &outParameterInfo);
void SetParameter(AudioUnitParameterID paramID, AudioUnitParameterValue value);
virtual OSStatus SetParameter(AudioUnitParameterID inID, AudioUnitScope inScope,
   AudioUnitElement inElement, AudioUnitParameterValue inValue, UInt32);
MidiControls* GetControls( MusicDeviceGroupID inChannel);
```

```
auUIObject

auUIObject(char* label, float* zone) : fLabel(label), fZone(zone);
virtual void GetName(char *text) ;
virtual void SetValue(double f) ;
virtual float GetValue() ;
virtual void GetDisplay(char *text);
virtual long GetID() ;
```

```
SynthNote

virtual bool Attack(const MusicDeviceNoteParams &inParams);
virtual void Kill(UInt32 inFrame);
virtual void Release(UInt32 inFrame);
virtual void FastRelease(UInt32 inFrame);
virtual Float32 Amplitude();
virtual OSStatus Render(UInt64 inAbsoluteSampleFrame, UInt32 i
```

```
AUEffectBase

Faust(AudioUnit component);
~Faust();
virtual OSStatus Version();
virtual OSStatus Initialize();
virtual OSStatus GetPropertyInfo(AudioUnitPropertyID inID, AudioUnitScope inScope, AudioUnitElement inElement, UInt32 & outDataSize, Boolean & outWritable);
virtual OSStatus GetProperty(AudioUnitPropertyID inID, AudioUnitScope inScope, AudioUnitElement inElement, void * outData);
virtual OSStatus GetParameterInfo(AudioUnitScope inScope, AudioUnitParameterID inParameterID, AudioUnitParameterInfo &outParameterInfo);
void SetParameter(AudioUnitParameterID paramID, AudioUnitParameterValue value);
virtual OSStatus SetParameter(AudioUnitParameterID inID, AudioUnitScope inScope, AudioUnitElement inElement, AudioUnitParameterValue inValue, UInt32);
virtual OSStatus GetPresets(CFArrayRef *outData) const;
virtual OSStatus NewFactoryPresetSet(const AUPreset & inNewFactoryPreset);
virtual bool SupportsTail() ;
virtual Float64 GetTailTime();
virtual Float64 GetLatency();
virtual OSStatus ProcessBufferLists(AudioUnitRenderActionFlags &ioActionFlags, const AudioBufferList &inBuffer, AudioBufferList &outBuffer, inFramesToProcess);
```

Figure 1 – UML Class diagram for the corresponding faust2au and faust2ausynth classes

5. MEASURING THE LATENCY

To measure the latency of, an impulse track was routed to two generated Audio Units, “Feedback Comb Filter with Delay”, and “Zita Reverb”. The zoomed out and zoomed in tracks in Audacity are shown in the following figures. The zoomed in figures are the snapshopts of the maximum zoom-in possible in Audacity. As illustrated, no latency can be observed in these figures, implying that it is less than 0.1 millisecond based on the shown time units.

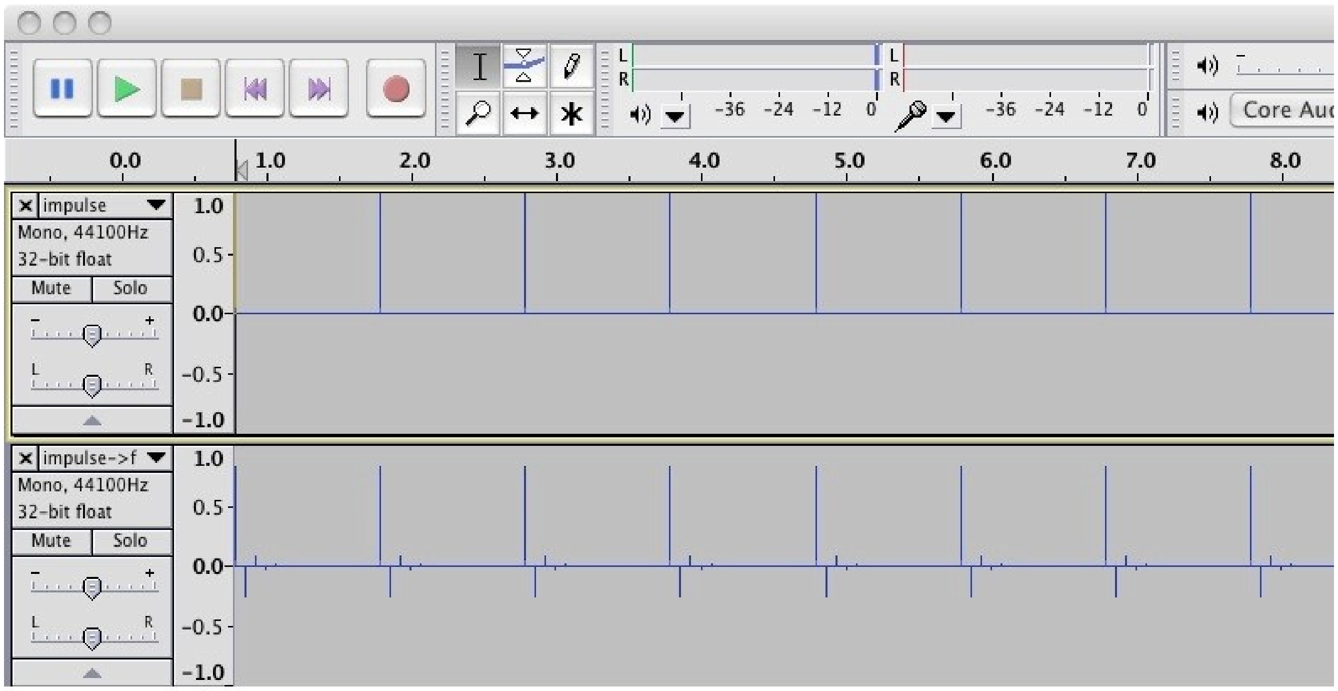


Figure 2 - Zoomed-out snapshot for “Feedback Comb Filter with Delay”

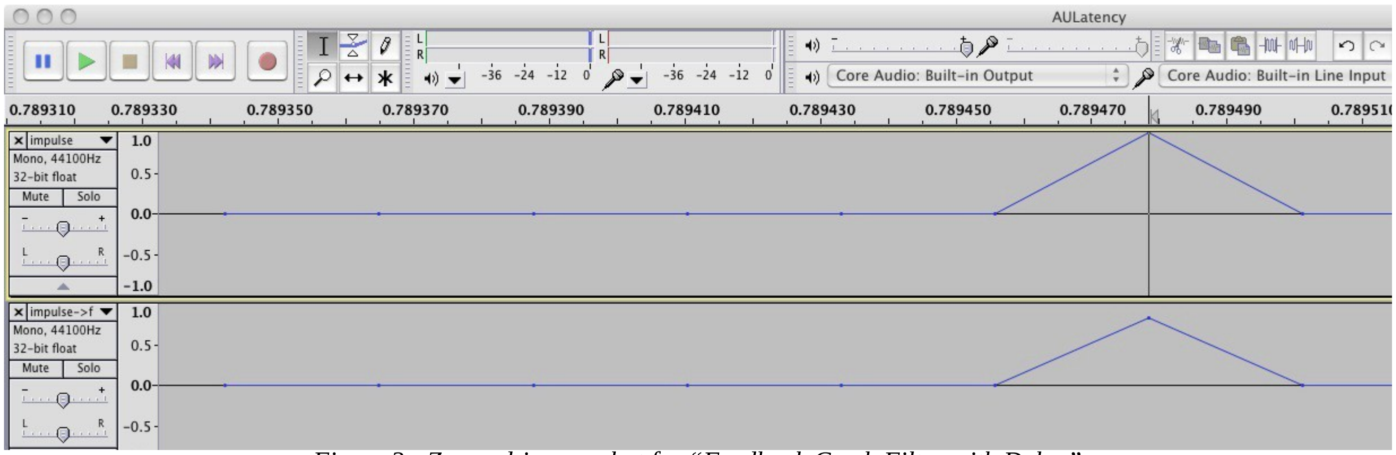


Figure 3 - Zoomed-in snapshot for “Feedback Comb Filter with Delay”

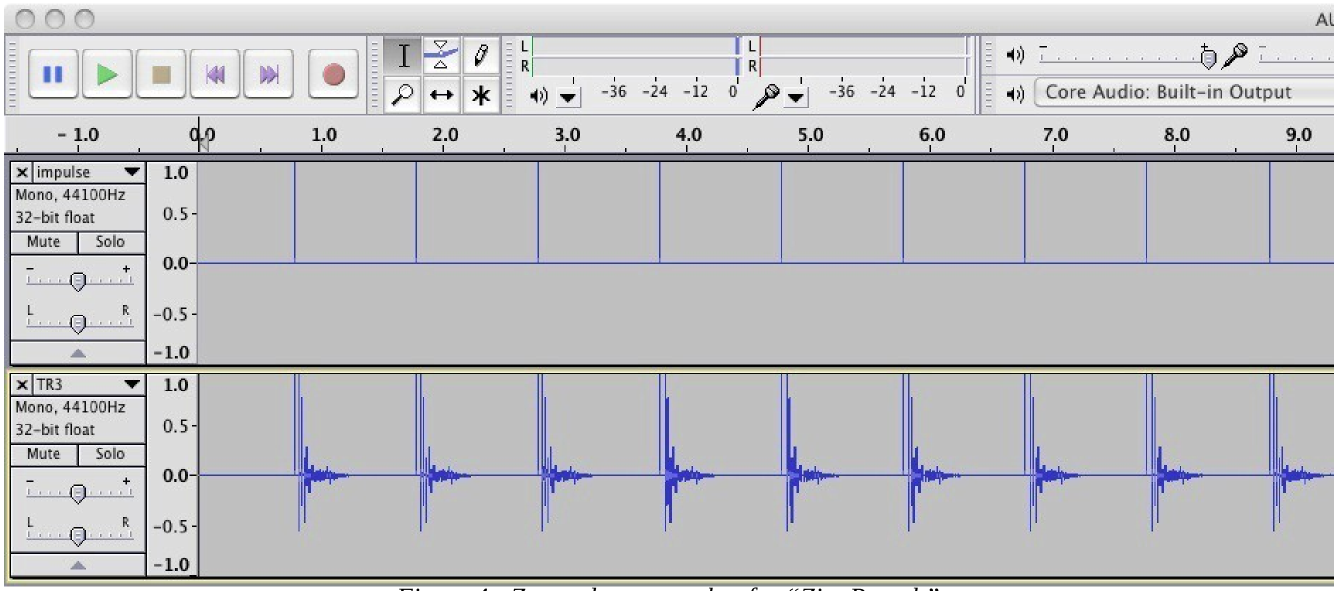


Figure 4 - Zoomed-out snapshot for "Zita Reverb"

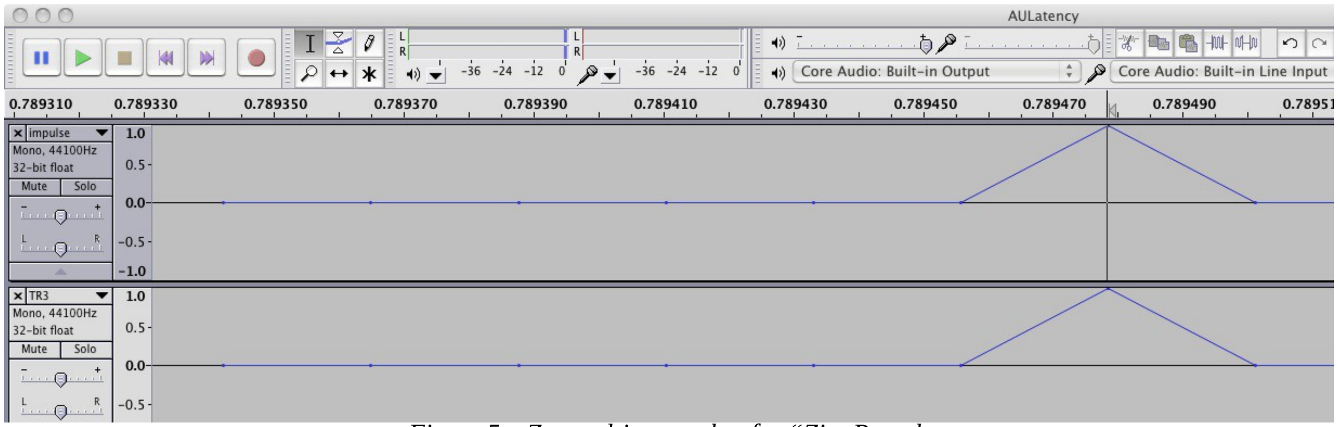


Figure 5 – Zoomed-in snapshot for "Zita Reverb

6. MAIN FILES

- `tools/faust2appls/faust2au`

  Bash shell script for gnerating AUEffect based on a faust dsp file
- `tools/faust2appls/faust2ausynth`

  Bash shell script for gnerating AUSynth based on a faust dsp file

- `architecture/au.cpp` – AUEffect architecture file
- `architecture/ausynth.cpp` – AUSynth architecture file

- `architecture/AU/Info.plist, architecture/AUSynth/Info.plist`

The information property list for AUEffect and AUSynth

- `architecture/AU/Info-template.plist, architecture/AUSynth/Info-template.plist`

  The information property list template, used by faust2au and faustausynth to generate AU by replacing some variables based in the command line arguments. The related variables include 'Name', 'Description', 'Subtype', 'Manufacturer', 'BundleId'

- `architecture/AU/FaustAU.xcodeproj, architecture/AUSynth/FaustAUSynth.xcodeproj`

  Xcode projects to be compiled by the bash scripts

- `architecture/AU/FaustAU.exp,, architecture/AUSynth/FaustAUSynth.exp` – Specify the entry points for running the AU component executable

- `architecture/AU/Source/AUSource/FaustAUVersion.h`

  `architecture/AUSynth/Source/AUSource/FaustAUSynthVersion.h`

  Specify the subtype, manufacturer and the version of the AudioUnit

- `architecture/AU/Source/AUSource/FaustAUVersion-template.h,`

  architecture/AUSynth/Source/AUSource/FaustAUSynthVersion-template.h

  Used by faust2au and faust2ausynth to generate AU by replacing some variables based in the command line arguments. The related variables include 'Subtype', 'Manufacturer'

- `architecture/AU/English.lproj/InfoPlist.strings, architecture/AUSynth/English.lproj/InfoPlist.strings`

  Specify 'BundleName', and 'BundleInfo' of the AUEffect and AUSynth

- `architecture/AU/English.lproj/InfoPlist-template.strings, architecture/AUSynth/English.lproj/InfoPlist-template.strings`

  The template files used by faust2au and faustausynth to generate AU by replacing some variables based in the command line arguments. The related variables include 'BundleName', 'BundleInfo'

- `architecture/AU/Source/AUSource/FaustAU.r, and architecture/AUSynth/Source/AUSource/FaustAUSynth.r`

  The resource files describing AUEffect and AUSynth properties

- `architecture/AU/Source/AUSource/FaustAU-template.r, and architecture/AUSynth/Source/AUSource/FaustAUSynth-template.r`

  The template files used by faust2au and faustausynth to generate AU by replacing some variables based in the command line arguments. The related variables include 'Name', and 'Description'

7. USAGE

The following is the description about the installation, command line arguments and the generated output related to both faust2au and faust2ausynth.

Installation

- copy the contents of the 'architecture' and 'tools' folders into the similar faust folders

- if needed, change FAUST_DIR variable in 'faust2au' (means faust2aueffect) and 'faust2ausynth' files in the 'tools/faust2appls/' folder

- in the terminal window, run 'install faust2au /usr/local/bin' and 'install faust2ausynth /usr/local/bin' (the files you just copied to 'faust2appls' in the previous step)

- Xcode should also be installed on your system (https://developer.apple.com/xcode/)

Running in Command Line

- with the default arguments:
  faust2au filename (or faust2ausynth filename)
  (e.g. faust2au echo.dsp)

- with some extra arguments:
  faust2au filename manufacturer STYP MANF

  'manufacturer' is the manufacturer of the AU (The default value is 'Grame')

  'STYP' is a four-character subtype (The default is the first four characters of the 'filename')

  'MANF' is a four-character manufacturer (The default is the first four characters of the 'manufacturer')

Output

- ~/Library/Audio/Plug-Ins/Components/'filename'.component

  AU Name will be: 'manufacturer: filename' ('filename' is used as the AU name).

  The names are case-sensitive, e.g. 'faust2au Echo.dsp' results in 'Echo' as the AU name

8. REFERENCES

Faust Quick Reference, Yann Orlarey, Grame, Centre National de Creation Musicale, April 2006
http://www.grame.fr/ressources/publications/faust-quick-reference.pdf

Signal Processing in Faust and PD, Julius Smith, REALSIMPLE Project, CCRMA
https://ccrma.stanford.edu/~jos/faust/faust.pdf

Audio Unit Programming Guide, Apple, 2007
https://developer.apple.com/library/mac/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/AudioUnitProgrammingGuide.pdf

Streinberg VST, http://www.steinberg.net/en/products/vst.html

Generating a VST Plugin via Faust, Julius Smith, CCRMA
https://ccrma.stanford.edu/~jos/faust/Generating_VST_Plugin_Faust.html