Faust iOS API

This API allows to interact with a Faust object and its associated audio engine on iOS at a high level. The idea is that all the audio part of the app is implemented in Faust allowing developers to focus on the design of the app itself.

For more details on how to create iOS apps from scratch using this tool, check the faust2api documentation or the Adding Faust Real-Time Audio Support to iOS Apps Tutorial.

Using This Package

This section is an accelerated version of the Adding Faust Real-Time Audio Support to iOS Apps Tutorial. We strongly recommend you to read it if this is the first time that you use this tool.

App Set-Up

Very little work has to be done to integrate this package to your iOS app.

First, in your app configuration in XCode, make sure that the AudioToolbox framework is imported in TARGETS/YouApp/BuildPhases/Link Binary With Libraries. If you used the -midi option when generating the API, you’ll also have to import the CoreMIDI framework.

Import DspFaust.h and DspFaust.cpp in your project (this can be done simply by dragging these files in your project tree). Then, import DspFaust.h (#import "DspFaust.h") in the file where you want to create/control the Faust object (e.g. your main ViewController). Make sure that the file where you import DspFaust.h has the .mm extension (this is necessary to be able to use C++ code in your objective-c file).

Using the API

The current Faust API is designed to seamlessly integrate to the life cycle of an iOS app. It is accessible through a single DspFaust object. The constructor of that object is used to set the sampling rate and the block size:

DspFaust *dspFaust = new DspFaust(SR,blockSize);

The start() method is used to start the audio computing and would typically be placed in the viewDidLoad method of the app’s main ViewController.

Similarly, stop() can be called to stop the audio computing and can be placed in didReceiveMemoryWarning along with the DspFaust destructor, etc.
It is possible to interact with the different parameters of the Faust object by using the `setParamValue` method. Two versions of this method exist: one where the parameter can be selected by its address and one where it can be selected using its ID. The Parameters List section gives a list of the addresses and corresponding IDs of the current Faust object.

If your Faust object is polyphonic (e.g. if you used the `-polyvoices` option when generating this API), then you can use the MIDI polyphony methods like `keyOn`, `keyOff`, etc.

It is possible to change the parameters of polyphonic voices independently using the `setVoiceParamValue` method. This method takes as one of its arguments the address to the voice returned by `keyOn` or `newVoice` when it is called. E.g:

```cpp
long voiceAddress = dspFaust->keyOn(70,100);
dspFaust->setVoiceParamValue(1,voiceAddress,214);
dspFaust->keyOff(70);
```

In the example above, a new note is created and its parameter ID 1 is modified. This note is then terminated. Note that parameters addresses (path) are different for independent voices than when using `setParamValue`. The list of these addresses is provided in a separate sub-section of the Parameters List section.

Finally, note that new voices don’t necessarily have to be created using `keyOn`. Indeed, you might choose to just use the `newVoice` method for that:

```cpp
long voiceAddress = dspFaust->newVoice;
dspFaust->setVoiceParamValue(1,voiceAddress,214);
dspFaust->deleteVoice(voiceAddress);
```

This is particularly useful when making apps where each finger of the user is an independent sound that doesn’t necessarily has a pitch.

**Parameters List**

**Main Parameters**

- **0**: `/Sequencer/DSP1/Polyphonic/Voices/Panic`
- **1**: `/Sequencer/DSP1/Polyphonic/Voices/synth/cutoff`
- **2**: `/Sequencer/DSP1/Polyphonic/Voices/synth/freq`
- **3**: `/Sequencer/DSP1/Polyphonic/Voices/synth/gain`
- **4**: `/Sequencer/DSP1/Polyphonic/Voices/synth/gate`
- **5**: `/Sequencer/DSP2/Zita_Rev1/Input/In_Delay`
- **6**: `/Sequencer/DSP2/Zita_Rev1/Decay_Times_in_Bands_(see_tooltips)/LF_X`
- **7**: `/Sequencer/DSP2/Zita_Rev1/Decay_Times_in_Bands_(see_tooltips)/Low_RT60`
- **8**: `/Sequencer/DSP2/Zita_Rev1/Decay_Times_in_Bands_(see_tooltips)/Mid_RT60`
- **9**: `/Sequencer/DSP2/Zita_Rev1/Decay_Times_in_Bands_(see_tooltips)/HF_Damping`
- **10**: `/Sequencer/DSP2/Zita_Rev1/RM_Peaking_Equalizer_1/Eq1_Freq`
Independent Voices

- 0: /synth/cutoff
- 1: /synth/freq
- 2: /synth/gain
- 3: /synth/gate

API Reference

DspFaust(int SR, int BS)

Constructor.

Arguments

- SR: sampling rate
- BS: block size

bool start()

Start the audio processing.
Returns true if successful and false if not.

void stop()

Stop the audio processing.

bool isRunning()

Returns true if audio is running.
long keyOn(int pitch, int velocity)

Instantiate a new polyphonic voice. This method can only be used if the [style:poly] metadata is used in the Faust code or if the -polyvoices flag has been provided before compilation.

keyOn will return 0 if the Faust object is not polyphonic or the address to the allocated voice as a long otherwise. This value can be used later with setVoiceParamValue or getVoiceParamValue to access the parameters of a specific voice.

Arguments
- pitch: MIDI note number (0-127)
- velocity: MIDI velocity (0-127)

int keyOff(int pitch)

De-instantiate a polyphonic voice. This method can only be used if the [style:poly] metadata is used in the Faust code or if the -polyvoices flag has been provided before compilation.

keyOff will return 0 if the object is not polyphonic and 1 otherwise.

Arguments
- pitch: MIDI note number (0-127), should be the same as the one used for keyOn

long newVoice()

Instantiate a new polyphonic voice. This method can only be used if the [style:poly] metadata is used in the Faust code or if -polyvoices flag has been provided before compilation.

keyOn will return 0 if the Faust object is not polyphonic or the address to the allocated voice as a long otherwise. This value can be used later with setVoiceParamValue, getVoiceParamValue or deleteVoice to access the parameters of a specific voice.
int deleteVoice(long voice)

De-instantiate a polyphonic voice. This method can only be used if the [style:poly] metadata is used in the Faust code or if --polyvoices flag has been provided before compilation.

deleteVoice will return 0 if the object is not polyphonic and 1 otherwise.

Arguments
- voice: the address of the voice given by newVoice

const char* getJSON()

Returns the JSON description of the Faust object.

int getParamsCount()

Returns the number of parameters of the Faust object.

void setParamValue(const char* address, float value)

Set the value of one of the parameters of the Faust object in function of its address (path).

Arguments
- address: address (path) of the parameter
  - value: value of the parameter

void setParamValue(int id, float value)

Set the value of one of the parameters of the Faust object in function of its id.
Arguments
- id: id of the parameter
- value: value of the parameter

float getParamValue(const char* address)

Returns the value of a parameter in function of its address (path).

Arguments
- address: address (path) of the parameter

float getParamValue(int id)

Returns the value of a parameter in function of its id.

Arguments
- id: id of the parameter

void setVoiceParamValue(const char* address, long voice, float value)

Set the value of one of the parameters of the Faust object in function of its address (path) for a specific voice.

Arguments
- address: address (path) of the parameter
- voice: address of the polyphonic voice (retrieved from keyOn
- value: value of the parameter

void setVoiceValue(int id, long voice, float value)

Set the value of one of the parameters of the Faust object in function of its id for a specific voice.
Arguments
• id: id of the parameter
• voice: address of the polyphonic voice (retrieved from keyOn
• value: value of the parameter

float getVoiceParamValue(const char* address, long voice)

Returns the value of a parameter in function of its address (path) for a specific voice.

Arguments
• address: address (path) of the parameter
• voice: address of the polyphonic voice (retrieved from keyOn)

float getVoiceParamValue(int id, long voice)

Returns the value of a parameter in function of its id for a specific voice.

Arguments
• id: id of the parameter
• voice: address of the polyphonic voice (retrieved from keyOn)

const char* getParamAddress(int id)

Returns the address (path) of a parameter in function of its ID.

Arguments
• id: id of the parameter

const char* getVoiceParamAddress(int id, long voice)

Returns the address (path) of a parameter in function of its ID.
Arguments
• id: id of the parameter
• voice: address of the polyphonic voice (retrieved from keyOn)

void propagateAcc(int acc, float v)
Propagate the RAW value of a specific accelerometer axis to the Faust object.

Arguments
• acc: the accelerometer axis (0: x, 1: y, 2: z)
• v: the RAW accelerometer value in m/s

void setAccConverter(int p, int acc, int curve, float amin, float amid, float amax)
Set the conversion curve for the accelerometer.

Arguments
• p: the UI parameter id
• acc: the accelerometer axis (0: x, 1: y, 2: z)
• curve: the curve (0: up, 1: down, 2: up and down)
• amin: mapping min point
• amid: mapping middle point
• amax: mapping max point

void propagateGyr(int gyr, float v)
Propagate the RAW value of a specific gyroscope axis to the Faust object.

Arguments
• gyr: the gyroscope axis (0: x, 1: y, 2: z)
• v: the RAW accelerometer value in m/s
void setGyrConverter(int p, int gyr, int curve, float amin, float amid, float amax)

Set the conversion curve for the gyroscope.

Arguments

- **p**: the UI parameter id
- **acc**: the accelerometer axis (0: x, 1: y, 2: z)
- **curve**: the curve (0: up, 1: down, 2: up and down)
- **amin**: mapping min point
- **amid**: mapping middle point
- **amax**: mapping max point

float getCPULoad()

Returns the CPU load.