FAUST Tutorial for Functional Programmers

Y. Orlarey, S. Letz, D. Fober, R. Michon

ICFP 2017 / FARM 2017
What is Faust?
What is Faust?

A programming language (DSL) to build electronic music instruments
Some Music Programming Languages

- 4CED
- Adagio
- AML
- AMPL
- Antescofo
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- *Chuck*
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- *Csound*
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- *Faust*
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- Keynote
- Kyma
- LOCO
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MUSIC 6
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musictex
- MUSIGOL
- MusicXML
- Musixtex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- *Puredata*
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- *Supercollider*
- Symbolic Composer
- Tidal
Brief Overview to Faust

- Faust offers end-users a high-level alternative to C to develop audio applications for a large variety of platforms.
- The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, Javascript, etc.).
- Faust is used on stage for concerts and artistic productions, for education and research, for open sources projects and commercial applications:
What Is Faust Used For?
Artistic Applications
Sonik Cube (Trafik/Orlarey), Smartfaust (Gracia), etc.
Open-Source projects

Guitarix: Hermann Meyer
Thanks to the HTML5/WebAudio API and Asm.js it is now possible to run synthesizers and audio effects from a simple web page!
Sound Spatialization
Ambitools: Pierre Lecomte, CNAM

**Ambitools** (Faust Award 2016), 3-D sound spatialization using Ambisonic techniques.
Medical Applications

Brain Stethoscope: Chris Chafe, CCRMA-Stanford

**Brain stethoscope** turns seizures into music in hopes of giving the listener an empathetic and intuitive understanding of the neural activity.
Simulation Applications
Stanford Car Simulator: Romain Michon, CCRMA-Stanford

**Stanford Car Simulator**, simulation of the sound of a car engine in Faust.
Simulation Applications
Bell simulations: Romain Michon and Chris Chafe, CCRMA-Stanford

CAD description of a bell turned into a procedural audio simulation in Faust
**Soniccloud**: every cell phone call can be perfectly calibrated for an individual’s unique Hearing Fingerprint – across 10 sonic dimensions.

The team at SonicCloud has had an outstanding experience working with Faust. Specifically, we have been able to optimize code to run our DSP algorithms in real-time without having to hand-optimize C/C++ code or write assembler. (Soniccloud)
Moforte (USA) designs musical instruments for iPad and iOS using Faust
Exercice 1: Djembe
Exercise 1: Djembe

Faust Online Compiler:
- https://faust.grame.fr/onlinecompiler

Faust Libraries Documentation:
- http://faust.grame.fr/libraries.html

Faust Code:

```plaintext
import("stdfaust.lib");

process = button("play")
    : pm.djembe(330, 0.8, 0.5, 1);
```
Faust Signals and Time Model
Faust Signals and Time Model

Faust programs operate on periodically sampled signals.

- A signal \( s \in S \) is a time to sample function.
- Two kinds of signals: \( S = S_Z \cup S_R \)
  - Integer signals: \( S_Z = \mathbb{Z} \rightarrow \mathbb{Z} \)
  - Floating-point signals: \( S_R = \mathbb{Z} \rightarrow \mathbb{R} \)
- The value of a Faust signal is always 0 before time 0:
  - \( \forall s \in S, s(t < 0) = 0 \)
- A Faust program denotes a signal processor \( p \in \mathbb{P} \), a (continuous) function that maps a group of \( n \) input signals to a group of \( m \) output signals:
  - \( \mathbb{P} = S^n \rightarrow S^m \)
Faust Signals and Time Model

Example, a "constant" signal

\[
y(t) = \begin{cases} 
1 & t \geq 0 \\
0 & t < 0 
\end{cases}
\]
Primitive Signal Processors
## Faust Primitives

### Arithmetic operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>$S^2 \rightarrow S^1$</td>
<td>addition: $y(t) = x_1(t) + x_2(t)$</td>
</tr>
<tr>
<td>-</td>
<td>$S^2 \rightarrow S^1$</td>
<td>subtraction: $y(t) = x_1(t) - x_2(t)$</td>
</tr>
<tr>
<td>*</td>
<td>$S^2 \rightarrow S^1$</td>
<td>multiplication: $y(t) = x_1(t) \times x_2(t)$</td>
</tr>
<tr>
<td>∧</td>
<td>$S^2 \rightarrow S^1$</td>
<td>power: $y(t) = x_1(t)^{x_2(t)}$</td>
</tr>
<tr>
<td>/</td>
<td>$S^2 \rightarrow S^1$</td>
<td>division: $y(t) = x_1(t)/x_2(t)$</td>
</tr>
<tr>
<td>%</td>
<td>$S^2 \rightarrow S^1$</td>
<td>modulo: $y(t) = x_1(t)% x_2(t)$</td>
</tr>
<tr>
<td>int</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cast into an int signal: $y(t) = (int)x(t)$</td>
</tr>
<tr>
<td>float</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cast into an float signal: $y(t) = (float)x(t)$</td>
</tr>
</tbody>
</table>
### Faust Primitives

#### Bitwise operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>logical AND: $y(t) = x_1(t) &amp; x_2(t)$</td>
</tr>
<tr>
<td></td>
<td>$S^2 \rightarrow S^1$</td>
<td>logical OR: $y(t) = x_1(t)</td>
</tr>
<tr>
<td>xor</td>
<td>$S^2 \rightarrow S^1$</td>
<td>logical XOR: $y(t) = x_1(t) \land x_2(t)$</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arith. shift left: $y(t) = x_1(t) &lt;&lt; x_2(t)$</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arith. shift right: $y(t) = x_1(t) &gt;&gt; x_2(t)$</td>
</tr>
</tbody>
</table>
Faust Primitives

Comparison operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>less than: $y(t) = x_1(t) &lt; x_2(t)$</td>
</tr>
<tr>
<td>&lt;=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>less or equal: $y(t) = x_1(t) \leq x_2(t)$</td>
</tr>
<tr>
<td>&gt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>greater than: $y(t) = x_1(t) &gt; x_2(t)$</td>
</tr>
<tr>
<td>&gt;=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>greater or equal: $y(t) = x_1(t) \geq x_2(t)$</td>
</tr>
<tr>
<td>==</td>
<td>$S^2 \rightarrow S^1$</td>
<td>equal: $y(t) = x_1(t) = x_2(t)$</td>
</tr>
<tr>
<td>!=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>different: $y(t) = x_1(t) \neq x_2(t)$</td>
</tr>
</tbody>
</table>
# Faust Primitives
## Trigonometric functions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc cosine: $y(t) = \text{acosf}(x(t))$</td>
</tr>
<tr>
<td>asin</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc sine: $y(t) = \text{asinf}(x(t))$</td>
</tr>
<tr>
<td>atan</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc tangent: $y(t) = \text{atanf}(x(t))$</td>
</tr>
<tr>
<td>atan2</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arc tangent of 2 signals: $y(t) = \text{atan2f}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>cos</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cosine: $y(t) = \text{cosf}(x(t))$</td>
</tr>
<tr>
<td>sin</td>
<td>$S^1 \rightarrow S^1$</td>
<td>sine: $y(t) = \text{sinf}(x(t))$</td>
</tr>
<tr>
<td>tan</td>
<td>$S^1 \rightarrow S^1$</td>
<td>tangent: $y(t) = \text{tanf}(x(t))$</td>
</tr>
</tbody>
</table>
# Faust Primitives

## Other Math operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>base-e exponential: ( y(t) = \expf(x(t)) )</td>
</tr>
<tr>
<td>log</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>base-e logarithm: ( y(t) = \logf(x(t)) )</td>
</tr>
<tr>
<td>log10</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>base-10 logarithm: ( y(t) = \log10f(x(t)) )</td>
</tr>
<tr>
<td>pow</td>
<td>( S^2 \rightarrow S^1 )</td>
<td>power: ( y(t) = \powf(x_1(t), x_2(t)) )</td>
</tr>
<tr>
<td>sqrt</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>square root: ( y(t) = \sqrtf(x(t)) )</td>
</tr>
<tr>
<td>abs</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>absolute value (int): ( y(t) = \text{abs}(x(t)) )</td>
</tr>
<tr>
<td>min</td>
<td>( S^2 \rightarrow S^1 )</td>
<td>minimum: ( y(t) = \text{min}(x_1(t), x_2(t)) )</td>
</tr>
<tr>
<td>max</td>
<td>( S^2 \rightarrow S^1 )</td>
<td>maximum: ( y(t) = \text{max}(x_1(t), x_2(t)) )</td>
</tr>
<tr>
<td>fmod</td>
<td>( S^2 \rightarrow S^1 )</td>
<td>float modulo: ( y(t) = \text{fmodf}(x_1(t), x_2(t)) )</td>
</tr>
<tr>
<td>remainder</td>
<td>( S^2 \rightarrow S^1 )</td>
<td>float remainder: ( y(t) = \text{remainderf}(x_1(t), x_2(t)) )</td>
</tr>
<tr>
<td>floor</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>largest int ( \leq ): ( y(t) = \text{floorf}(x(t)) )</td>
</tr>
<tr>
<td>ceil</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>smallest int ( \geq ): ( y(t) = \text{ceilf}(x(t)) )</td>
</tr>
<tr>
<td>rint</td>
<td>( S^1 \rightarrow S^1 )</td>
<td>closest int: ( y(t) = \text{rintf}(x(t)) )</td>
</tr>
</tbody>
</table>
# Faust Primitives

## Delays and Tables

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mem</strong></td>
<td>$S^1 \rightarrow S^1$</td>
<td>1-sample delay: $y(t + 1) = x(t), y(0) = 0$</td>
</tr>
<tr>
<td>@</td>
<td>$S^2 \rightarrow S^1$</td>
<td>delay: $y(t + x_2(t)) = x_1(t), y(t &lt; x_2(t)) = 0$</td>
</tr>
<tr>
<td><strong>rdtable</strong></td>
<td>$S^3 \rightarrow S^1$</td>
<td>read-only table: $y(t) = T[r(t)]$</td>
</tr>
<tr>
<td><strong>rwtable</strong></td>
<td>$S^5 \rightarrow S^1$</td>
<td>read-write table: $T[w(t)] = c(t); y(t) = T[r(t)]$</td>
</tr>
<tr>
<td><strong>select2</strong></td>
<td>$S^3 \rightarrow S^1$</td>
<td>select between 2 signals: $T[] = {x_0(t), x_1(t)}; y(t) = T[s(t)]$</td>
</tr>
<tr>
<td><strong>select3</strong></td>
<td>$S^4 \rightarrow S^1$</td>
<td>select between 3 signals: $T[] = {x_0(t), x_1(t), x_2(t)}; y(t) = T$</td>
</tr>
</tbody>
</table>
# Faust Primitives

## User Interface Primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>button(str)</code></td>
<td><code>button(&quot;play&quot;)</code></td>
</tr>
<tr>
<td><code>checkbox(str)</code></td>
<td><code>checkbox(&quot;mute&quot;)</code></td>
</tr>
<tr>
<td><code>vslider(str, cur, min, max, inc)</code></td>
<td><code>vslider(&quot;vol&quot;,50,0,100,1)</code></td>
</tr>
<tr>
<td><code>hslider(str, cur, min, max, inc)</code></td>
<td><code>hslider(&quot;vol&quot;,0.5,0,1,0.01)</code></td>
</tr>
<tr>
<td><code>nentry(str, cur, min, max, inc)</code></td>
<td><code>nentry(&quot;freq&quot;,440,0,8000,1)</code></td>
</tr>
<tr>
<td><code>vgroup(str, block-diagram)</code></td>
<td><code>vgroup(&quot;reverb&quot;, ...)</code></td>
</tr>
<tr>
<td><code>hgroup(str, block-diagram)</code></td>
<td><code>hgroup(&quot;mixer&quot;, ...)</code></td>
</tr>
<tr>
<td><code>tgroup(str, block-diagram)</code></td>
<td><code>vgroup(&quot;parametric&quot;, ...)</code></td>
</tr>
<tr>
<td><code>vbargraph(str, min, max)</code></td>
<td><code>vbargraph(&quot;input&quot;,0,100)</code></td>
</tr>
<tr>
<td><code>hbargraph(str, min, max)</code></td>
<td><code>hbargraph(&quot;signal&quot;,0,1.0)</code></td>
</tr>
</tbody>
</table>
Exercice 2: Adding rhythm and sliders to the Djembe
Exercise 2: Adding rhythm to the Djembe

Faust Code:

```faust
import("stdfaust.lib");

process = button("play"), // try checkbox("play")
    ba.pulsen(100, 4800) : *
    : pm.djembe(330, 0.8, 0.5, 1);
```
Exercise 3: Adding sliders to the Djembe

Faust Code:

```faust
import("stdfaust.lib");

process = checkbox("play"),
    ba.pulsen(100, 4800) : *
    : pm.djembe(
        hslider("freq", 300, 100, 1000, 1),
        hslider("position", 0.8, 0, 1, 0.1),
        hslider("sharpness", 0.5, 0, 1, 0.1),
        hslider("gain", 0.5, 0, 1, 0.1)
    );
```
Exercise 4: Adding an echo to the Djembe

Faust Code:

```faust
import("stdfaust.lib");

echo = +~(@(22100):*(hslider("fb",0,0,1,0.01)));

process = checkbox("play"),
    ba.pulsen(100, 4800) : *
    : pm.djembe(
        hslider("freq", 300, 100, 1000, 1),
        hslider("position", 0.8, 0, 1, 0.1),
        hslider("sharpness", 0.5, 0, 1, 0.1),
        hslider("gain",0.5,0,1,0.1)
    )
    : echo;
```
Programming by composition
Programming by Composition

Block-Diagram Algebra

Programming by patching is familiar to musicians:
Programming by Composition

Block-Diagram Algebra

Programming by patching, the ENIAC computer:
Programming by Composition

Block-Diagram Algebra

Block-diagrams are widely used in Visual Programming Languages like Max/MSP:
Programming by Composition

Block-Diagram Algebra

Faust allows structured block-diagrams, here part of the zita reverb.

```
allpass_combs(8) feedbackmatrix(8) delayfilters(...1, 8, 0.1))
fbdelaylines(8) zita_rev_fdn(...1, 8, 0.1))))(48000)
```
Programming by Composition

Composition Operations

- \((A<:B)\) split composition (associative, priority 1)
- \((A:>B)\) merge composition (associative, priority 1)
- \((A:B)\) sequential composition (associative, priority 2)
- \((A,B)\) parallel composition (associative, priority 3)
- \((A^B)\) recursive composition (priority 4)
Programming by Composition
Same Expression in Lambda-Calculus, FP and Faust

Lambda-Calculus
\x.\y.(x+y,x*y) 2 3

FP/FL (John Backus)
[+,*]::<2,3>

Faust
2,3 <: +,*

Figure: block-diagram of 2,3 <: +,*
Programming by Composition
A Very Simple Example

process = 1 : +~_;

\[
y(t) = \begin{cases} 
0 & t < 0 \\
1 + y(t - 1) & t \geq 0
\end{cases}
\]
The *parallel composition* \((A,B)\) is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

\[
(A,B): (S^n \rightarrow S^m) \rightarrow (S^{n'} \rightarrow S^{m'}) \rightarrow (S^{n+n'} \rightarrow S^{m+m'})
\]

**Figure:** Example of parallel composition \((10, \ast)\)
Programming by Composition

Sequential Composition (associative, priority 2)

The *sequential composition* \((A : B)\) connects the outputs of \(A\) to the corresponding inputs of \(B\).

\[
(A : B) : (S^n \rightarrow S^m) \rightarrow (S^m \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)
\]

**Figure:** Example of sequential composition \(((*,/) : +)\)
Programming by Composition

Split Composition (associative, priority 1)

The *split composition* $(A <: B)$ operator is used to distribute the outputs of $A$ to the inputs of $B$

$$(A <: B): (S^n \rightarrow S^m) \rightarrow (S^{k.m} \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)$$

**Figure:** example of split composition $((10,20) <: (+,*,/))$
Programming by Composition

Merge Composition (associative, priority 1)

The *merge composition* \((A :> B)\) is used to connect several outputs of \(A\) to the same inputs of \(B\). Signals connected to the same input are added.

\[(A :> B): (S^n \rightarrow S^{k.m}) \rightarrow (S^m \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)\]

*Figure:* example of merge composition \(((10, 20, 30, 40) :> *)\)
Programming by Composition

Recursive Composition (priority 4)

The *recursive composition* $(A \sim B)$ is used to create cycles in the block-diagram in order to express recursive computations.

$$(A \sim B): (S^n + n' \rightarrow S^m + m') \rightarrow (S^m' \rightarrow S^{n'}) \rightarrow (S^n \rightarrow S^m + m')$$

*Figure: example of recursive composition $+(12345) \sim *(1103515245)$*
Faust Program
A Faust program is essentially a list of statements. These statements can be:

- metadata declarations,
- file imports
- definitions

Example:

```faust
declare name "noise";
declare copyright "(c)GRAME 2006";
import("music.lib");
process = noise * vslider("volume", 0, 0, 1, 0.1);
```
A definition associates an identifier with an expression it stands for. Example:

```
random = +(12345) ~ *(1103515245);
```
Definitions

Functions' definitions

```
definition

identifier ( parameter ) = expression ;
```

- Definitions with formal parameters correspond to functions' definitions. Example:
  
  ```
  linear2db(x) = 20*log10(x);
  ```

- Alternative notation using a *lambda-abstraction*:
  
  ```
  linear2db = \(x).(20*log10(x));
  ```
**Definitions**

**Pattern Matching Definitions**

*definition*

- identifier -> ( pattern ) = expression ;

- Formal parameters can also be full expressions representing patterns. Example:

  ```
  duplicate(1,exp) = exp;
  duplicate(n,exp) = exp, duplicate(n-1,exp);
  ```

- Alternative notation:

  ```
  duplicate = case {
    (1,exp) => exp;
    (n,exp) => duplicate(n-1,exp);
  };
  ```
statement

import file

fileimport

- allows to import definitions from other source files.
- for example `import("math.lib");` imports the definitions from "math.lib" file, a set of additional mathematical functions provided as foreign functions.
Each Faust expression has an associated *lexical environment*
Environments

With Expression

```
withexpression
```

- With expression allows to specify a *local environment*, a private list of definitions that will be used to evaluate the left hand expression.
- Example pink noise filter:

```plaintext
pink = f : + ~ g with {
  f(x) = 0.04957526213389*x
  - 0.06305581334498*x@1
  + 0.01483220320740*x@2;
  g(x) = 1.80116083982126*x
  - 0.80257737639225*x@1;
};
```
Environments

Environment

\begin{definition}
\textit{environment}
\end{definition}

- an \textbf{environment} is used to group together related definitions:

\begin{verbatim}
constant = environment {
   pi = 3.14159;
   e = 2.718 ;
   ....
};
\end{verbatim}

- definitions of an environment can be easily accessed: \texttt{constant.pi}
Environments

Library

*library*

- allows to create an environment by reading the definitions from a file.
- example: `library("filter.lib")`
- definitions are accessed like this: `library("filter.lib").smooth`
Environments

Component

$\textit{component}$

- allows to reuse a full Faust program as a simple expression.
- example:

  $\text{component("osc.dsp")::<component("freeverb.dsp")}$

- equivalence between:

  $\text{component("freeverb.dsp")}$

  and

  $\text{library("freeverb.dsp").process}$
Expressions

Iterations

\textit{diagiteration}

- Iterations are analog to \texttt{for(\ldots)} loops
- provide a convenient way to automate some complex block-diagram constructions.
Expressions

Iterations

The following example shows the use of `seq` to create a 10-bands filter:

```plaintext
define process = seq(i, 10,
  vgroup("band\%i",
    bandfilter( 1000*(1+i) )
  ),
);
Exercise 5: Djembe automation
Exercise 5: Djembe automation

Faust Code:

```faust
import("stdfaust.lib");

saw(f) = f/ma.SR : (+,1:fmod)~_;

process = checkbox("play"),
    ba.pulsen(100, 4800) : *
    : pm.djembe(
        hslider("freq", 300, 100, 1000, 1),
        saw(hslider("fpos",1,0.05,20,0.01)),
        saw(hslider("fsharp",1,0.05,20,0.01)),
        saw(hslider("fgain",1,0.05,20,0.01))
    )
;
```
Faust Ecosystem
Faust Ecosystem

Overview

- **Faust Ecosystem**
  - **Faust** command line compiler (lin, osx)
  - faust2puredata
  - faust2max6
  - faust2vst
  - faust2android
  - faust2ios
  - ... command line builders

- FaustWorks IDE (lin, osx)
- **FaustLive** IDE with embedded Faust compiler (lin, osx, win)
- **faustgen~** embedded Faust compiler for max (osx, win)
- **faustcompile** embedded Faust compiler for csound6 (lin, osx, win)
- **faust4processing** embedded Faust compiler for Processing (lin, osx, win)

- **Embedded Compiler**

- **Web Servers**
  - http://faustservice.grame.fr/
    - Compiler API (used by FaustLive)
  - http://faust.grame.fr/index.php/online-examples
    - Online IDE (any browser)
  - http://faust.grame.fr/www/faustplayground
    - Simplified Faust programming
Faust Ecosystem

Command-line Tools

Simplify the compilation workflow: full automated process to build Android and iOS applications, VST plugins, Max/MSP externals, Csound opcodes, etc.
Faust Ecosystem

FaustWorks

FaustWorks can simplify Faust learning in particular by providing "realtime" code and diagram generation:
Faust Ecosystem

FaustLive

FaustLive speeds up the Edit/Compile/Run. It provides advanced cooperation features:
Faust Ecosystem

Faustgen

Faustgen speeds up the Edit/Compile/Run within the Max framework:

```
declare name "fst_cub~
declare version "1.33"
// declare author "Julius O. Smith"
// declare license "STL-4.3"
import("math.lib");
import("music.lib");
import("effect.lib");
import("filter.lib");

// ----------------------------- cubicnl(drive,offset) -----------------------------
// Cubic nonlinearity distortion
//
// USAGE: cubicnl(drive,offset), where
// drive = distortion amount, between 0 and 1
// offset = constant added before nonlinearity to give even harmonics
// Note: offset can introduce a nonzero mean - Feed
// cubicnl output to dbcocker to remove this.

// REFERENCES:
// https://ccrma.stanford.edu/~jos/pasp/Nonlinear_Distortion.html

// direct from effect lib and combine with a filter. A dc remover will be in
sf1 = vslider("freq-low-cut",130,20,1000,1):smooth(0.99);
sf2 = vslider("freq-high-cut",5000,1000,10000,1):smooth(0.99);

drive = vslider("drive",0,0,1,0.01);
offset = vslider("offset",0,0,1,0.01);

process = cubicnl(drive,offset); speaker(s1,s2); mixcha;
```
Faust4processing provides an embedded Faust compiler for Processing:

```java
void setup()
{
    // prepare the user interface
    size(480, 120);
    cps = new ControlPS(this);
    System.out.println("Working Directory = " + System.getProperty("user.dir"));

    // Define and compile osc signal processor
    osc = new FaustProcessing(this, "osc";

    "// Sinusoidal Oscillator"
    "Sinusoidal Oscillator"
    "n +
    "import ("music.lib"; \n"
    "n +
    "db2linear(x) = pow(10.0, x/20.0); \n"
    "n +
    "smooth(c) = (1-c) \n"
    "vol = hslider("volume [unit:db]", -20, -96, 0, 0.1) : db2linear :
    "freq = hslider("freq [unit:Hz]", 1000, 26, 24000, 1);

    "process = vgroup("Oscillator", osc(freq + vol));\n"

    "// build osc's user interface
    int nbr=osc.getParamsCount();
    size(600, 50 + nbr*25 + 45);
    cps = new ControlPS(this);
    for (int i=0; i<nbr; i++) {
        cps.addSlider(osc.get("i")
            setPosition(30, i*25+50)
            setSize(300, 20)
            .setRange(osc.getParamMin(i), osc.getParamMax(i))
            .setValue(osc.getParamValue(i))
    }
```
PMIX speeds up the Edit/Compile/Run within VST host:
Faust Ecosystem

Online compiler

The Online compiler can be used from a web browser to compile Faust programs for a variety of systems, including the Web.

http://faust.grame.fr/onlinecompiler
Faust Ecosystem

Faust playground

The Web as a gigantic Lego box to reuse and recompose audio applications. http://faust.grame.fr/faustplayground
Thanks! Questions?