INSTITUTE OF SONOLOGY

Royal Conservatoire The Hague

SuperCollider

Composing with Algorithms
http://www.bjarni-gunnarsson.net

SuperCollider

SuperCollider

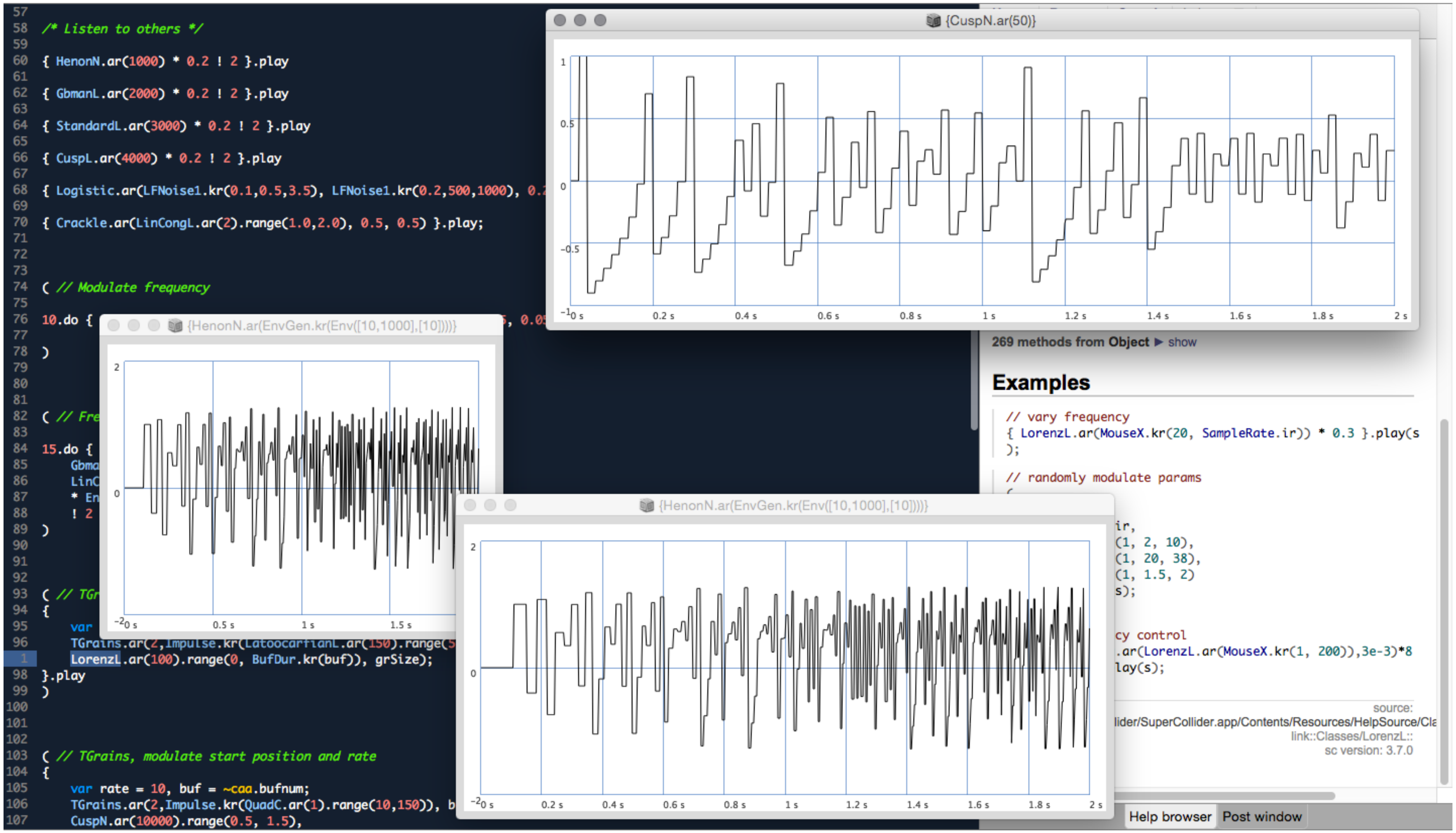
SuperCollider is an environment for *real time audio* and *composition*.

SuperCollider consists of an interpreted **object-oriented language** and a state of the art **realtime sound synthesis server**.

SuperCollider supports different activities such as sound synthesis, digital signal processing, algorithmic composition, live electronics and live coding.

SuperCollider is open source and free software, released under the terms of the GNU General Public License

SuperCollider


{CuspN.ar(50)}
{HenonN.ar(EnvGen.kr(Env([10,1000],[10])))}
{HenonN.ar(EnvGen.kr(Env([10,1000],[10])))}
Help browser
Post window

```
/* Listen to others */

{ HenonN.ar(1000) * 0.2 ! 2 }.play

{ GbmanL.ar(2000) * 0.2 ! 2 }.play

{ StandardL.ar(3000) * 0.2 ! 2 }.play

{ CuspL.ar(4000) * 0.2 ! 2 }.play

{ Logistic.ar(LFNoise1.kr(0.1,0.5,3.5), LFNoise1.kr(0.2,500,1000), 0.

{ Crackle.ar(LinCongL.ar(2).range(1.0,2.0), 0.5, 0.5) }.play;



( // Modulate frequency

10.do {                                                          , 0.05

)



( // Fre

15.do {
    Gbma
    LinC
    * En
    ! 2
)



( // TGr
{
    var
    TGrains.ar(2,Impulse.kr(LatoocarfianL.ar(150).range(5
    LorenzL.ar(100).range(0, BufDur.kr(buf)), grSize);
}.play
)



( // TGrains, modulate start position and rate
{
    var rate = 10, buf = ~caa.bufnum;
    TGrains.ar(2,Impulse.kr(QuadC.ar(1).range(10,150)), b
    CuspN.ar(10000).range(0.5, 1.5),
```

269 methods from Object ▶ show

Examples

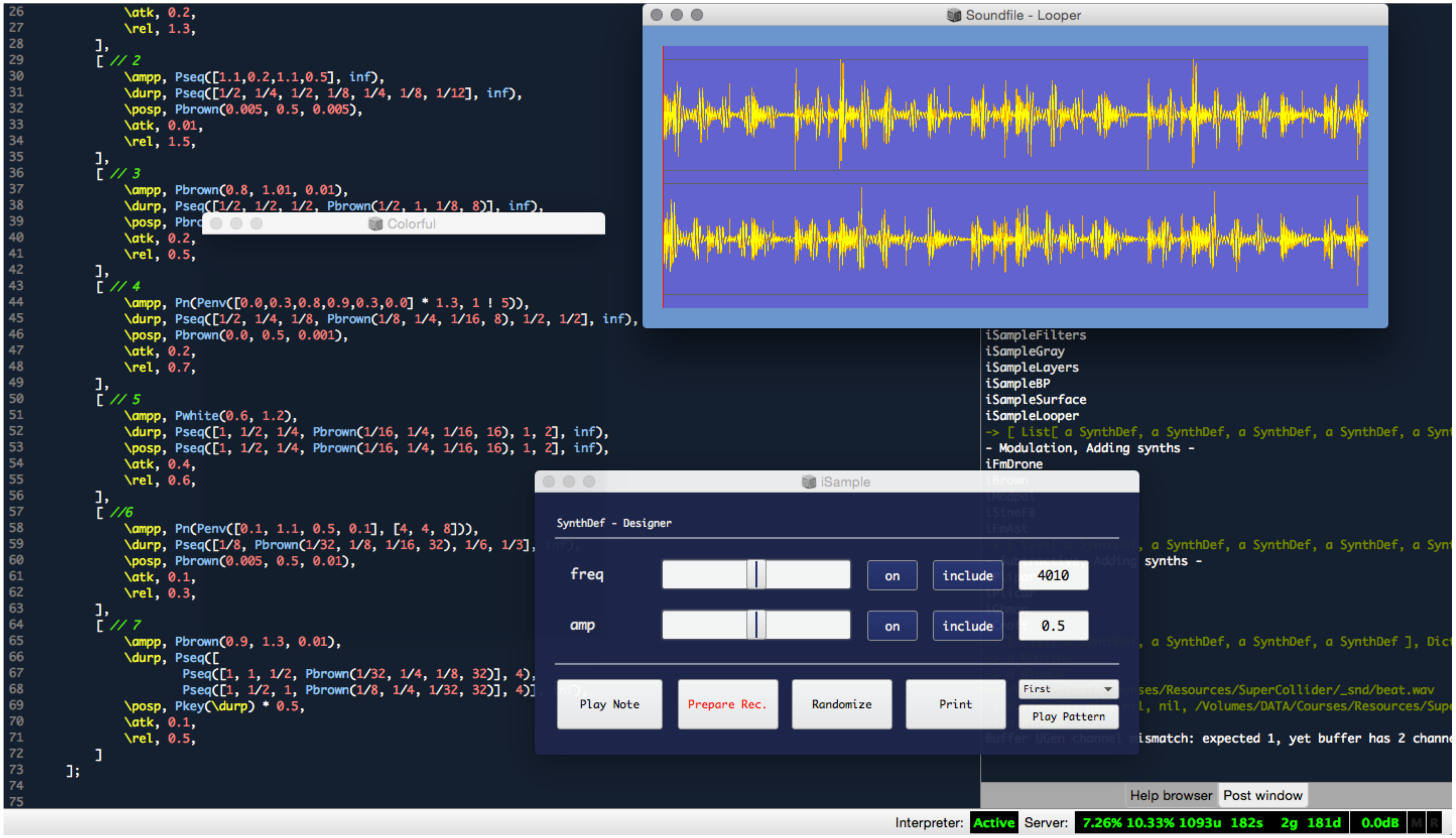
```
// vary frequency
{ LorenzL.ar(MouseX.kr(20, SampleRate.ir)) * 0.3 }.play(s
);

// randomly modulate params
(
                 ir,
                 (1, 2, 10),
                 (1, 20, 38),
                 (1, 1.5, 2)
                 s);

                 cy control
                 .ar(LorenzL.ar(MouseX.kr(1, 200)),3e-3)*8
                 lay(s);
```

source:
lider/SuperCollider.app/Contents/Resources/HelpSource/Cla
link::Classes/LorenzL::
sc version: 3.7.0

SuperCollider



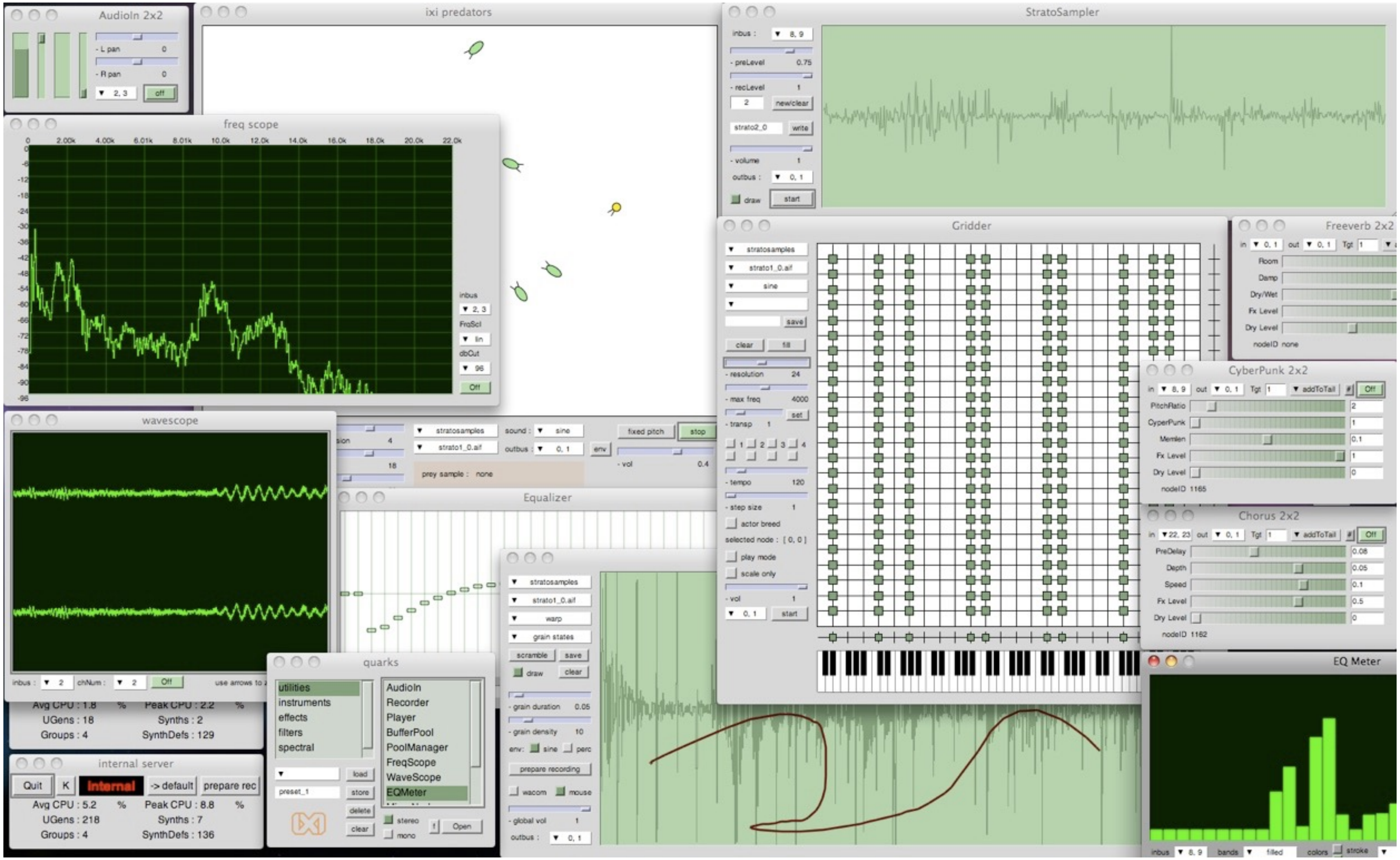
```
            \atk, 0.2,
            \rel, 1.3,
        ],
        [ // 2
            \ampp, Pseq([1.1,0.2,1.1,0.5], inf),
            \durp, Pseq([1/2, 1/4, 1/2, 1/8, 1/4, 1/8, 1/12], inf),
            \posp, Pbrown(0.005, 0.5, 0.005),
            \atk, 0.01,
            \rel, 1.5,
        ],
        [ // 3
            \ampp, Pbrown(0.8, 1.01, 0.01),
            \durp, Pseq([1/2, 1/2, 1/2, Pbrown(1/2, 1, 1/8, 8)], inf),
            \posp, Pbro
            \atk, 0.2,
            \rel, 0.5,
        ],
        [ // 4
            \ampp, Pn(Penv([0.0,0.3,0.8,0.9,0.3,0.0] * 1.3, 1 ! 5)),
            \durp, Pseq([1/2, 1/4, 1/8, Pbrown(1/8, 1/4, 1/16, 8), 1/2, 1/2], inf),
            \posp, Pbrown(0.0, 0.5, 0.001),
            \atk, 0.2,
            \rel, 0.7,
        ],
        [ // 5
            \ampp, Pwhite(0.6, 1.2),
            \durp, Pseq([1, 1/2, 1/4, Pbrown(1/16, 1/4, 1/16, 16), 1, 2], inf),
            \posp, Pseq([1, 1/2, 1/4, Pbrown(1/16, 1/4, 1/16, 16), 1, 2], inf),
            \atk, 0.4,
            \rel, 0.6,
        ],
        [ //6
            \ampp, Pn(Penv([0.1, 1.1, 0.5, 0.1], [4, 4, 8])),
            \durp, Pseq([1/8, Pbrown(1/32, 1/8, 1/16, 32), 1/6, 1/3], inf),
            \posp, Pbrown(0.005, 0.5, 0.01),
            \atk, 0.1,
            \rel, 0.3,
        ],
        [ // 7
            \ampp, Pbrown(0.9, 1.3, 0.01),
            \durp, Pseq([
                Pseq([1, 1, 1/2, Pbrown(1/32, 1/4, 1/8, 32)], 4),
                Pseq([1, 1/2, 1, Pbrown(1/8, 1/4, 1/32, 32)], 4)], inf),
            \posp, Pkey(\durp) * 0.5,
            \atk, 0.1,
            \rel, 0.5,
        ]
    ];
```

freq: 4010
amp: 0.5

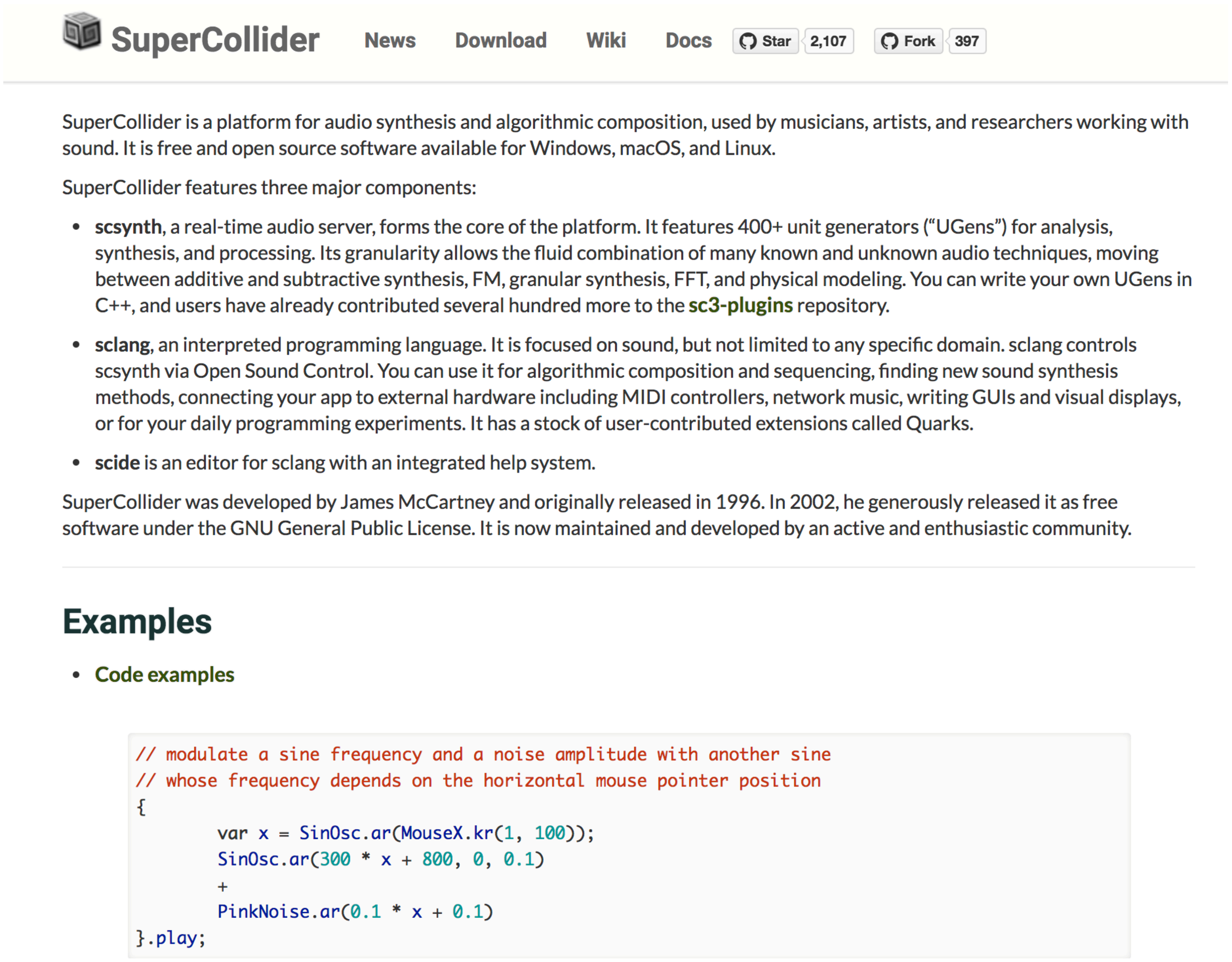
```
iSampleFilters
iSampleGray
iSampleLayers
iSampleBP
iSampleSurface
iSampleLooper
-> [ List[ a SynthDef, a SynthDef, a SynthDef, a SynthDef, a Synt
- Modulation, Adding synths -
iFmDrone
iBrown
iModpal
iSineFB
iFmAst
-> [ List[ a SynthDef, a SynthDef, a SynthDef, a SynthDef, a Synt
- Subtractive, Adding synths -
iPlipar
iPlicar
iChmon
iCroft
-> [ List[ a SynthDef, a SynthDef, a SynthDef, a SynthDef ], Dict
-> a Routine
-> a Window
-> /Volumes/DATA/Courses/Resources/SuperCollider/_snd/beat.wav
-> Buffer(0, nil, nil, nil, /Volumes/DATA/Courses/Resources/Supe
-> a Window
Buffer UGen channel mismatch: expected 1, yet buffer has 2 channe
```

Interpreter: Active Server: 7.26% 10.33% 1093u 182s 2g 181d 0.0dB M R

SuperCollider



IXI Quarks



SuperCollider is a platform for audio synthesis and algorithmic composition, used by musicians, artists, and researchers working with sound. It is free and open source software available for Windows, macOS, and Linux.

SuperCollider features three major components:

- **scsynth**, a real-time audio server, forms the core of the platform. It features 400+ unit generators ("UGens") for analysis, synthesis, and processing. Its granularity allows the fluid combination of many known and unknown audio techniques, moving between additive and subtractive synthesis, FM, granular synthesis, FFT, and physical modeling. You can write your own UGens in C++, and users have already contributed several hundred more to the **sc3-plugins** repository.
- **sclang**, an interpreted programming language. It is focused on sound, but not limited to any specific domain. sclang controls scsynth via Open Sound Control. You can use it for algorithmic composition and sequencing, finding new sound synthesis methods, connecting your app to external hardware including MIDI controllers, network music, writing GUIs and visual displays, or for your daily programming experiments. It has a stock of user-contributed extensions called Quarks.
- **scide** is an editor for sclang with an integrated help system.

SuperCollider was developed by James McCartney and originally released in 1996. In 2002, he generously released it as free software under the GNU General Public License. It is now maintained and developed by an active and enthusiastic community.

Examples

- **Code examples**

```
// modulate a sine frequency and a noise amplitude with another sine
// whose frequency depends on the horizontal mouse pointer position
{
	var x = SinOsc.ar(MouseX.kr(1, 100));
	SinOsc.ar(300 * x + 800, 0, 0.1)
	+
	PinkNoise.ar(0.1 * x + 0.1)
}.play;
```

http://supercollider.github.io

Design Goals

To realize sound processes that are **different** every time they are played.

To write pieces in a way that **describes a range of possibilities** rather than a fixed entity

To facilitate **live improvisation** by a composer/performer.

(McCartney, Rethinking the Computer Music Language: SuperCollider)

About

The SuperCollider **language** is based on *Smalltalk* and is used for creating **programs** that communicate with the **synthesis server** in order to make sounds.

Unit Generators (UGens) are used for generating and processing audio signals within the synthesis server. Interconnected *UGens* are packaged into a **SynthDef** that describes which UGens are used and how they connect.

The **objects** of the SuperCollider language objects together data and methods that act on that data.

About

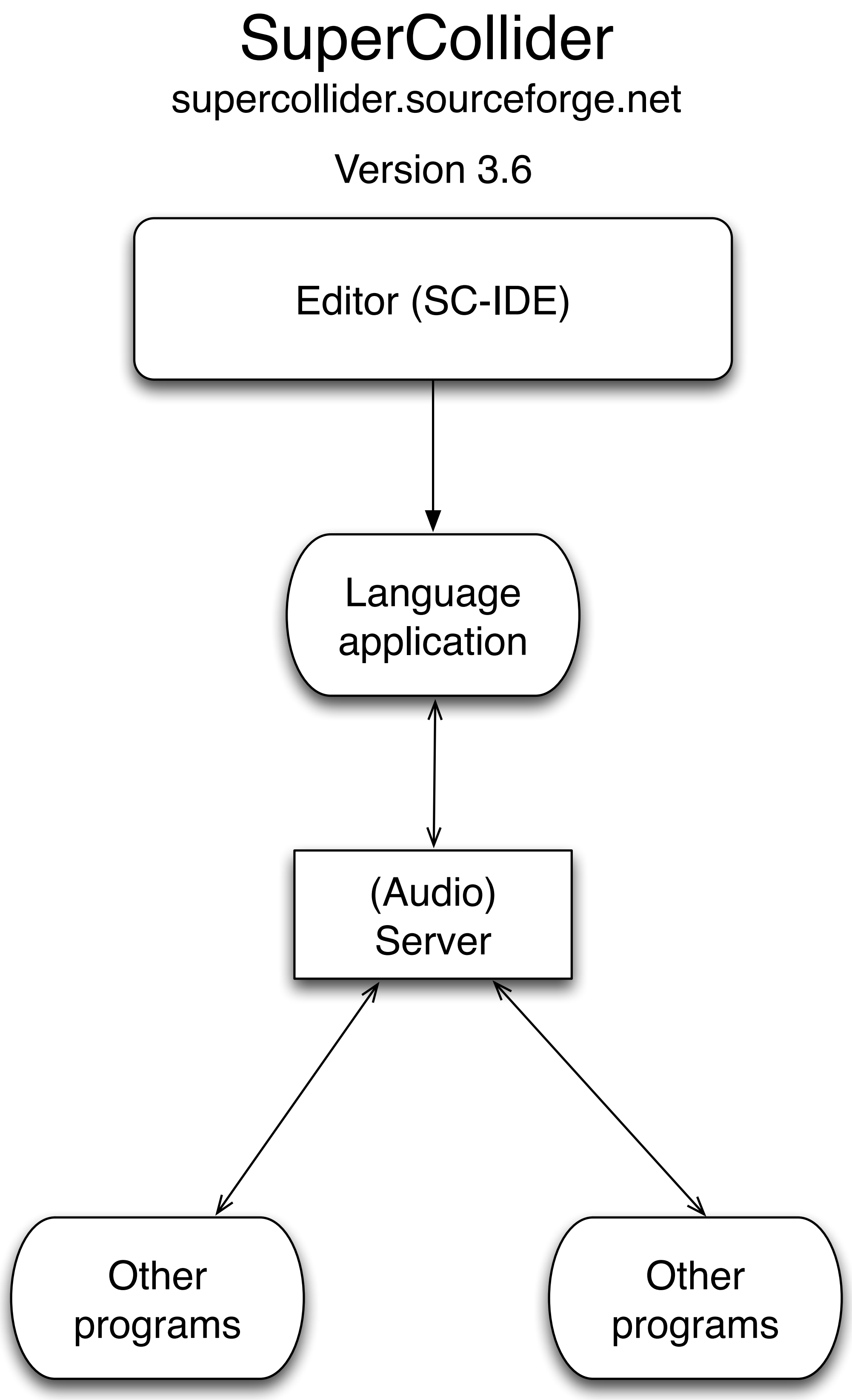
Classes describe *attributes* and *behavior* that objects have in common.

Sounds that are played or transformed are stored in **buffers** that exist inside the synthesis server.

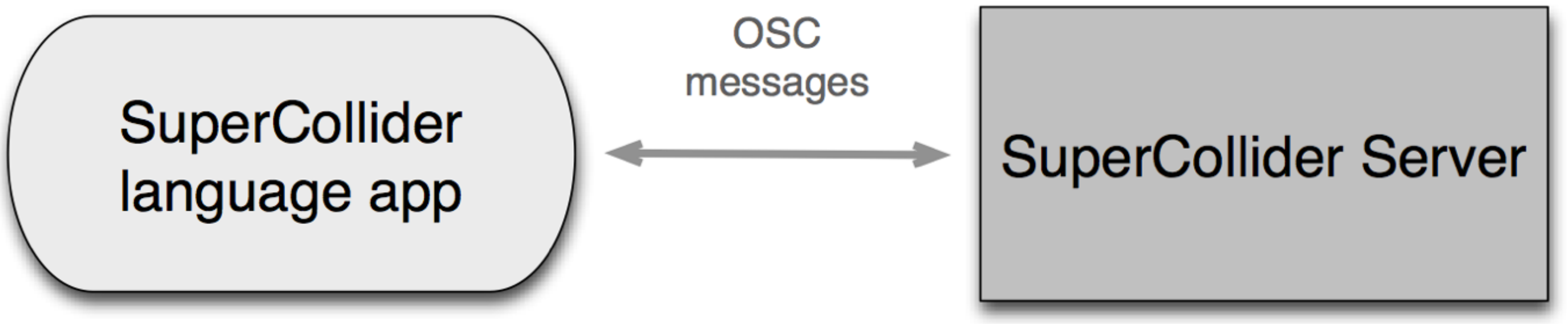
Audio channels that SuperCollider synths use for sending their sound through are called audio **buses**.

Compositional logic and scheduling is implemented with **routines**, **tasks** and **patterns**.

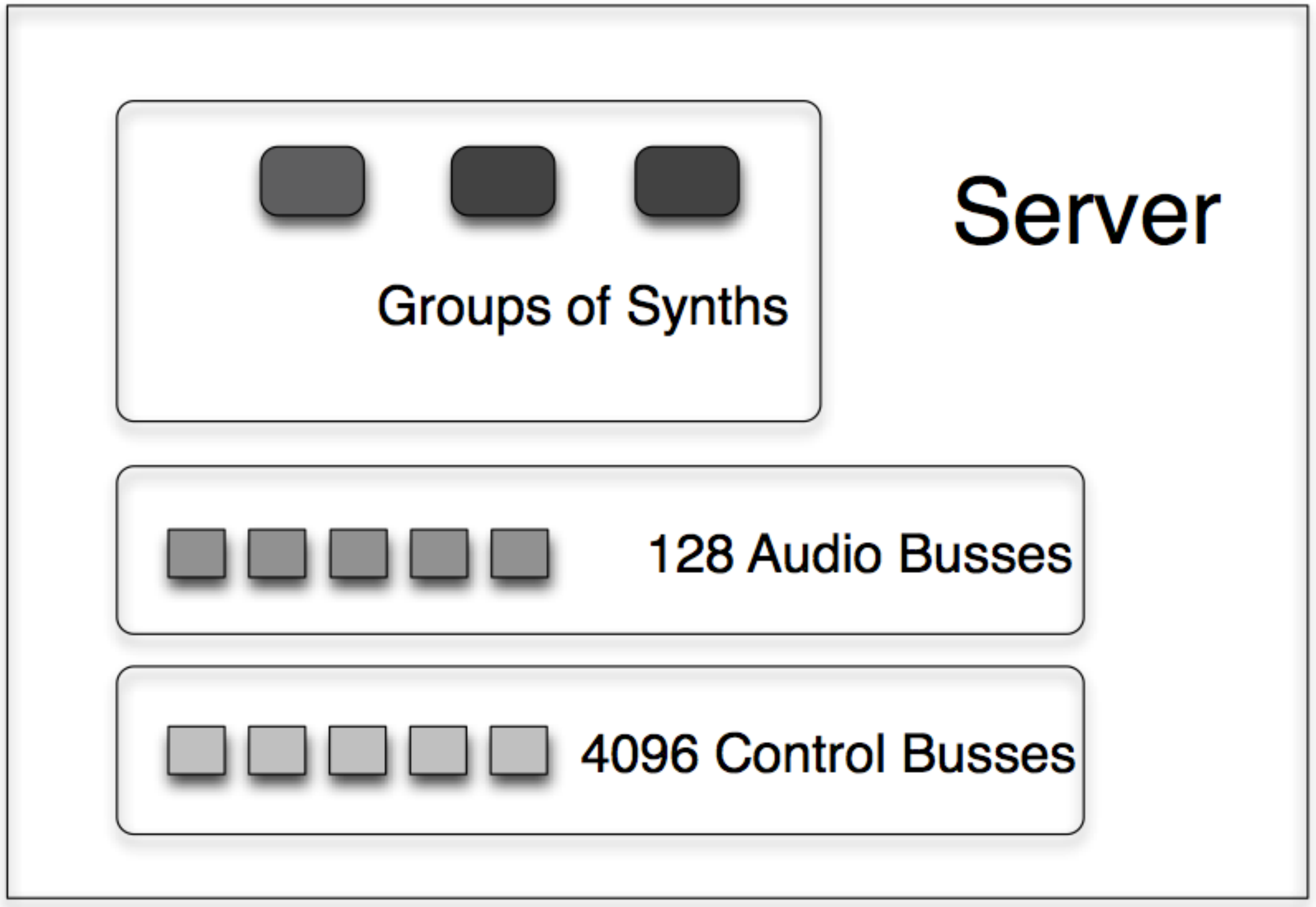
Architecture



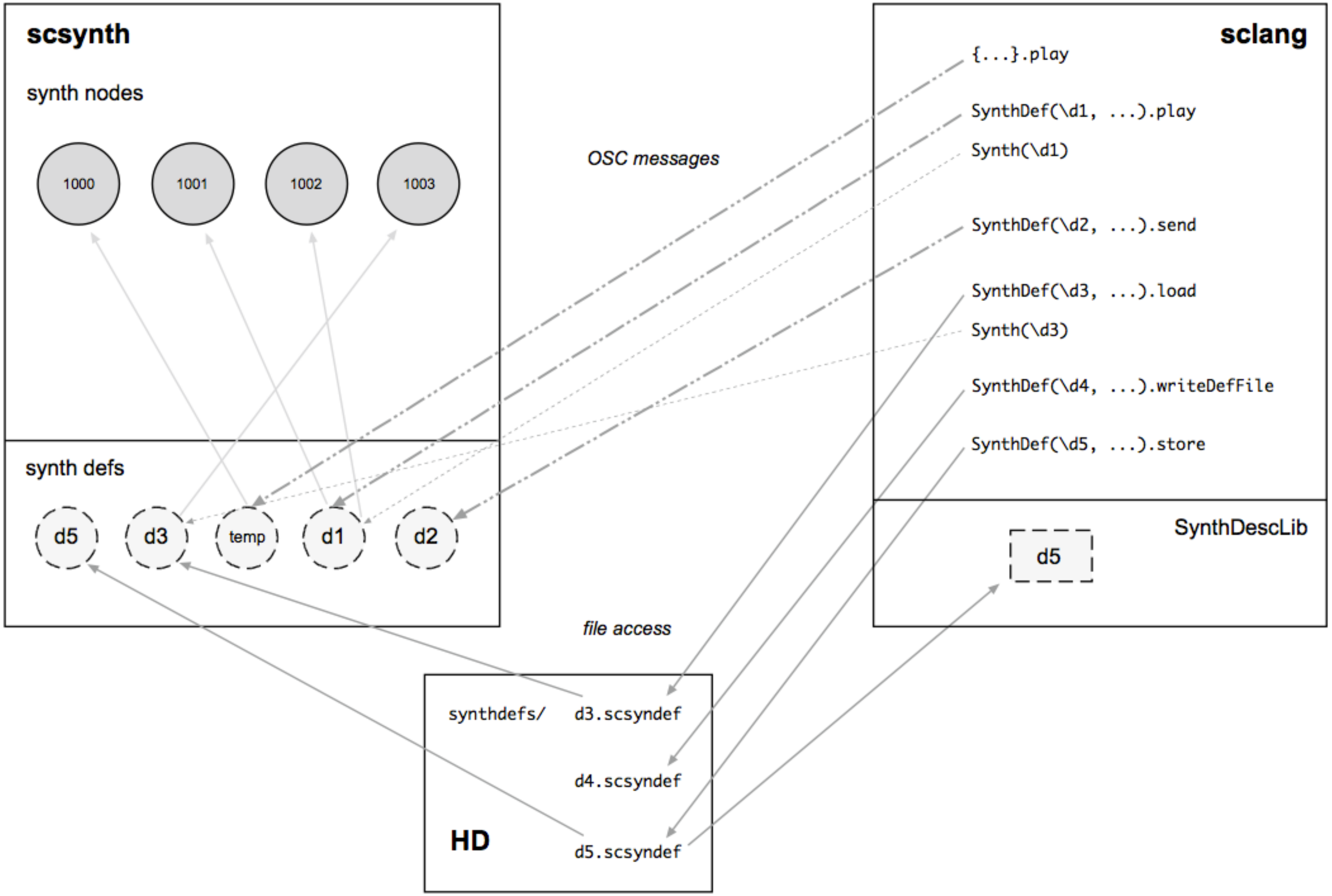
Messaging


SuperCollider language app
OSC messages
SuperCollider Server

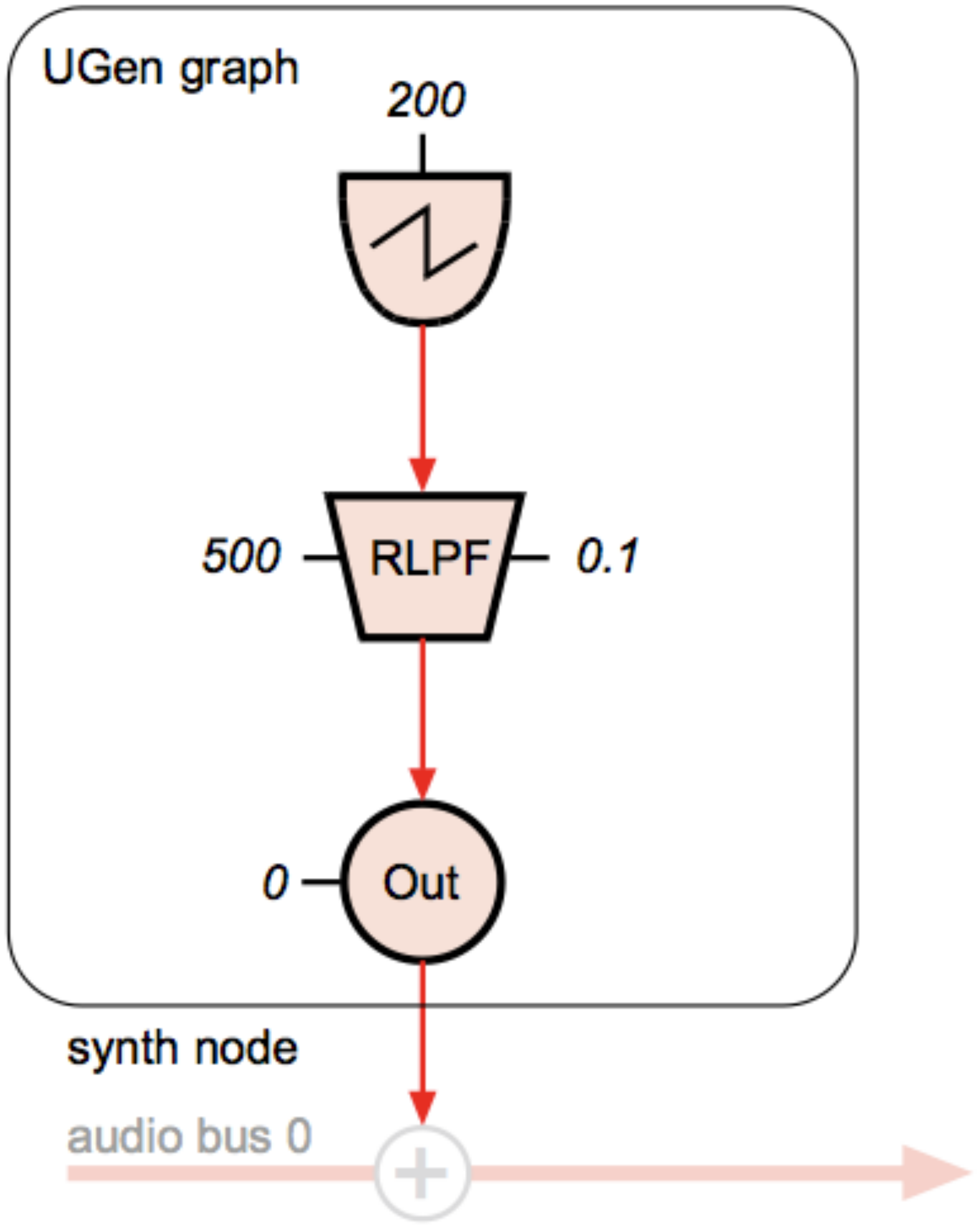
Architecture



Architecture



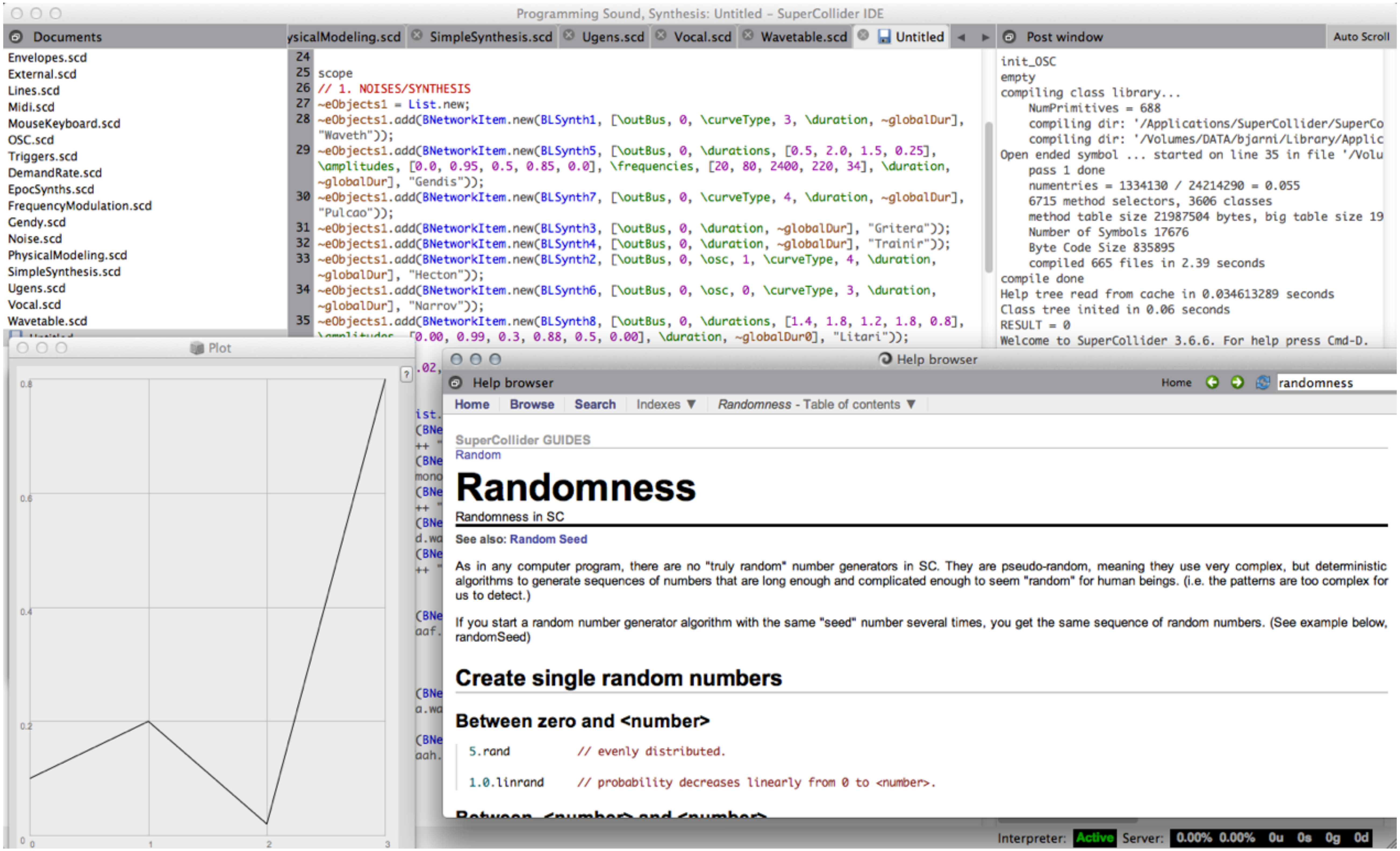
Synthesis Graphs



Synth Nodes and UGen graphs

```
(
SynthDef("simple", {
var sig;
sig = Saw.ar(200);
sig = RLPF.ar(sig, 500, 0.1);
Out.ar(0, sig);
}).play;
)
```

IDE



Programming Sound, Synthesis: Untitled – SuperCollider IDE

Documents

```
Envelopes.scd
External.scd
Lines.scd
Midi.scd
MouseKeyboard.scd
OSC.scd
Triggers.scd
DemandRate.scd
EpocSynths.scd
FrequencyModulation.scd
Gendy.scd
Noise.scd
PhysicalModeling.scd
SimpleSynthesis.scd
Ugens.scd
Vocal.scd
Wavetable.scd
```

ysicalModeling.scd | SimpleSynthesis.scd | Ugens.scd | Vocal.scd | Wavetable.scd | Untitled

```
scope
// 1. NOISES/SYNTHESIS
~eObjects1 = List.new;
~eObjects1.add(BNetworkItem.new(BLSynth1, [\outBus, 0, \curveType, 3, \duration, ~globalDur],
"Waveth"));
~eObjects1.add(BNetworkItem.new(BLSynth5, [\outBus, 0, \durations, [0.5, 2.0, 1.5, 0.25],
\amplitudes, [0.0, 0.95, 0.5, 0.85, 0.0], \frequencies, [20, 80, 2400, 220, 34], \duration,
~globalDur], "Gendis"));
~eObjects1.add(BNetworkItem.new(BLSynth7, [\outBus, 0, \curveType, 4, \duration, ~globalDur],
"Pulcao"));
~eObjects1.add(BNetworkItem.new(BLSynth3, [\outBus, 0, \duration, ~globalDur], "Gritera"));
~eObjects1.add(BNetworkItem.new(BLSynth4, [\outBus, 0, \duration, ~globalDur], "Trainir"));
~eObjects1.add(BNetworkItem.new(BLSynth2, [\outBus, 0, \osc, 1, \curveType, 4, \duration,
~globalDur], "Hecton"));
~eObjects1.add(BNetworkItem.new(BLSynth6, [\outBus, 0, \osc, 0, \curveType, 3, \duration,
~globalDur], "Narrov"));
~eObjects1.add(BNetworkItem.new(BLSynth8, [\outBus, 0, \durations, [1.4, 1.8, 1.2, 1.8, 0.8],
\amplitudes, [0.00, 0.99, 0.3, 0.88, 0.5, 0.00], \duration, ~globalDur0], "Litari"));
```

Post window — Auto Scroll

```
init_OSC
empty
compiling class library...
    NumPrimitives = 688
    compiling dir: '/Applications/SuperCollider/SuperCo
    compiling dir: '/Volumes/DATA/bjarni/Library/Applic
Open ended symbol ... started on line 35 in file '/Volu
    pass 1 done
    numentries = 1334130 / 24214290 = 0.055
    6715 method selectors, 3606 classes
    method table size 21987504 bytes, big table size 19
    Number of Symbols 17676
    Byte Code Size 835895
    compiled 665 files in 2.39 seconds
compile done
Help tree read from cache in 0.034613289 seconds
Class tree inited in 0.06 seconds
RESULT = 0
Welcome to SuperCollider 3.6.6. For help press Cmd-D.
```

Help browser — Home — randomness

Home | Browse | Search | Indexes ▼ | *Randomness* - Table of contents ▼

SuperCollider GUIDES
Random

Randomness

Randomness in SC

See also: Random Seed

As in any computer program, there are no "truly random" number generators in SC. They are pseudo-random, meaning they use very complex, but deterministic algorithms to generate sequences of numbers that are long enough and complicated enough to seem "random" for human beings. (i.e. the patterns are too complex for us to detect.)

If you start a random number generator algorithm with the same "seed" number several times, you get the same sequence of random numbers. (See example below, randomSeed)

Create single random numbers

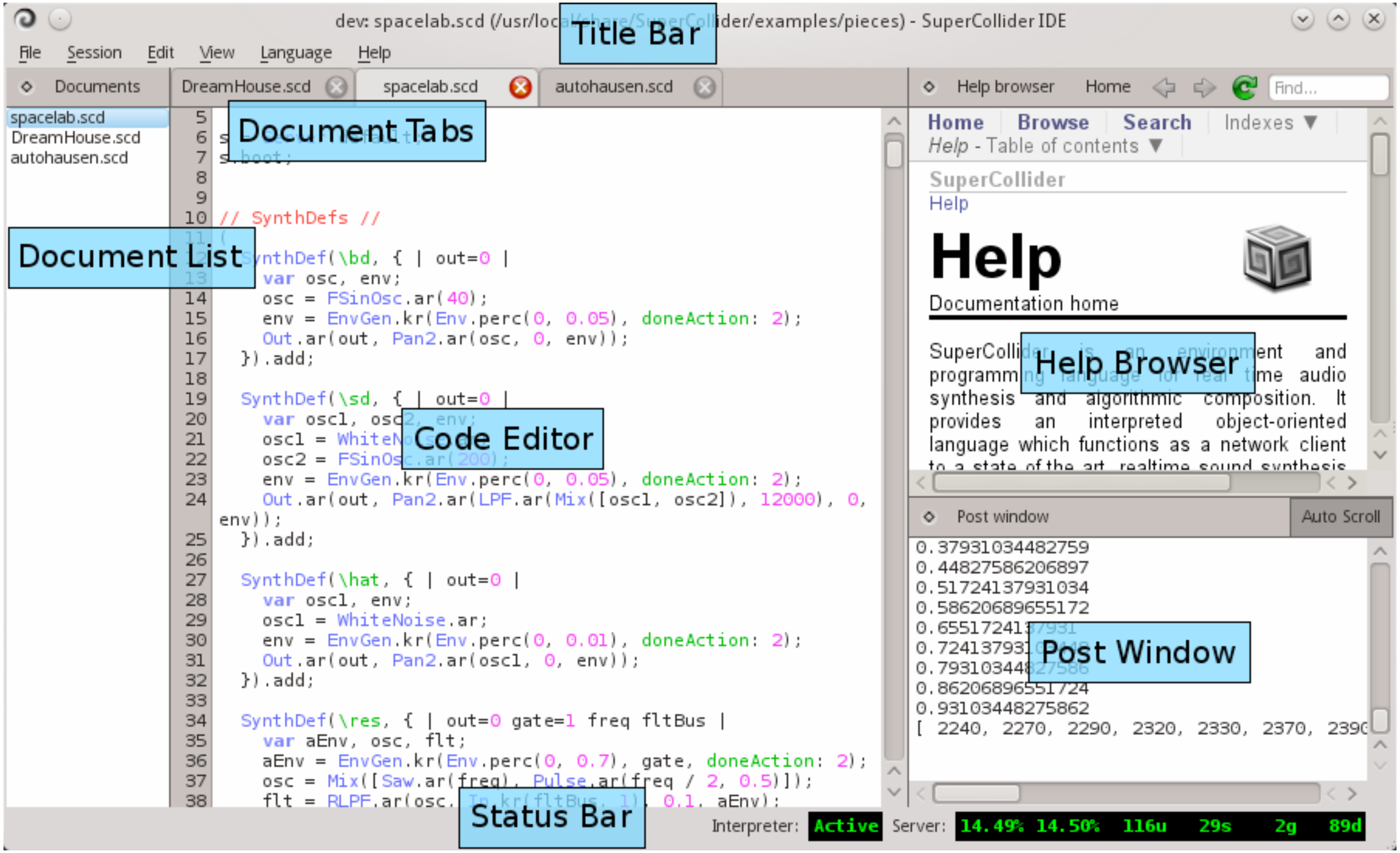
Between zero and <number>

```
5.rand          // evenly distributed.

1.0.linrand     // probability decreases linearly from 0 to <number>.
```

Interpreter: Active   Server: 0.00% 0.00% 0u 0s 0g 0d

IDE


dev: spacelab.scd (/usr/local/share/SuperCollider/examples/pieces) - SuperCollider IDE
Title Bar
File Session Edit View Language Help
Documents
DreamHouse.scd
spacelab.scd
autohausen.scd
Document Tabs
Document List
Code Editor
Help browser
Home
Find...
Home Browse Search Indexes ▼
Help - Table of contents ▼
Help Browser
Post window
Auto Scroll
Post Window
Status Bar
Interpreter: Active
Server: 14.49% 14.50% 116u 29s 2g 89d

```
s = Server.default;
s.boot;


// SynthDefs //
(
SynthDef(\bd, { | out=0 |
    var osc, env;
    osc = FSinOsc.ar(40);
    env = EnvGen.kr(Env.perc(0, 0.05), doneAction: 2);
    Out.ar(out, Pan2.ar(osc, 0, env));
  }).add;

  SynthDef(\sd, { | out=0 |
    var osc1, osc2, env;
    osc1 = WhiteNoise.ar;
    osc2 = FSinOsc.ar(200);
    env = EnvGen.kr(Env.perc(0, 0.05), doneAction: 2);
    Out.ar(out, Pan2.ar(LPF.ar(Mix([osc1, osc2]), 12000), 0,
env));
  }).add;

  SynthDef(\hat, { | out=0 |
    var osc1, env;
    osc1 = WhiteNoise.ar;
    env = EnvGen.kr(Env.perc(0, 0.01), doneAction: 2);
    Out.ar(out, Pan2.ar(osc1, 0, env));
  }).add;

  SynthDef(\res, { | out=0 gate=1 freq fltBus |
    var aEnv, osc, flt;
    aEnv = EnvGen.kr(Env.perc(0, 0.7), gate, doneAction: 2);
    osc = Mix([Saw.ar(freq), Pulse.ar(freq / 2, 0.5)]);
    flt = RLPF.ar(osc, In.kr(fltBus, 1), 0.1, aEnv);
```

SuperCollider
Help

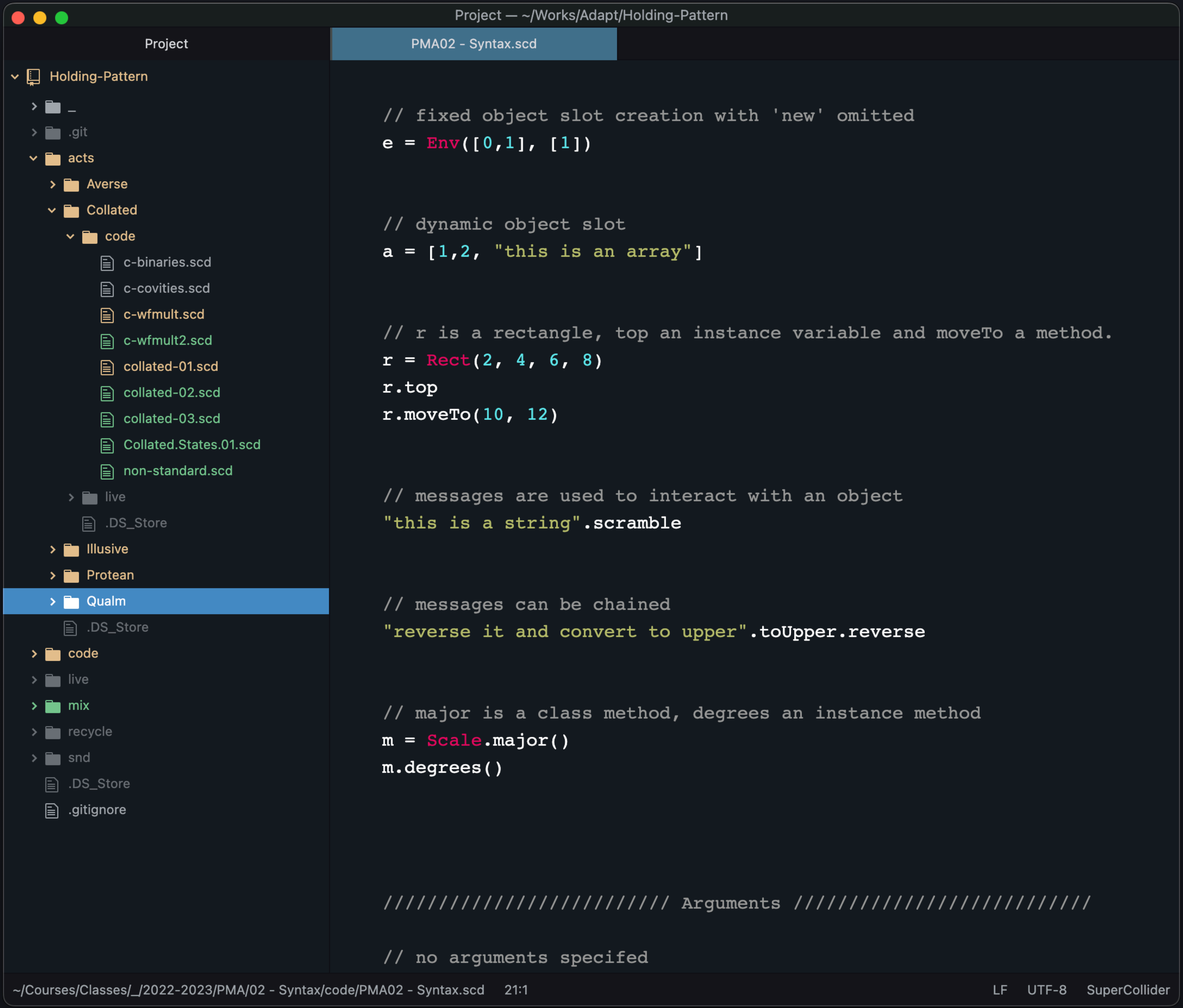
Help

Documentation home

SuperCollider is an environment and programming language for real time audio synthesis and algorithmic composition. It provides an interpreted object-oriented language which functions as a network client to a state of the art realtime sound synthesis

```
0.37931034482759
0.44827586206897
0.51724137931034
0.58620689655172
0.65517241379931
0.72413793103448
0.79310344827586
0.86206896551724
0.93103448275862
[ 2240, 2270, 2290, 2320, 2330, 2370, 2390
```

Pulsar



Project — ~/Works/Adapt/Holding-Pattern

Project | PMA02 - Syntax.scd

- Holding-Pattern
  - _
  - .git
  - acts
    - Averse
    - Collated
      - code
        - c-binaries.scd
        - c-covities.scd
        - c-wfmult.scd
        - c-wfmult2.scd
        - collated-01.scd
        - collated-02.scd
        - collated-03.scd
        - Collated.States.01.scd
        - non-standard.scd
      - live
      - .DS_Store
    - Illusive
    - Protean
    - Qualm
    - .DS_Store
  - code
  - live
  - mix
  - recycle
  - snd
  - .DS_Store
  - .gitignore

```
// fixed object slot creation with 'new' omitted
e = Env([0,1], [1])


// dynamic object slot
a = [1,2, "this is an array"]


// r is a rectangle, top an instance variable and moveTo a method.
r = Rect(2, 4, 6, 8)
r.top
r.moveTo(10, 12)


// messages are used to interact with an object
"this is a string".scramble


// messages can be chained
"reverse it and convert to upper".toUpper.reverse


// major is a class method, degrees an instance method
m = Scale.major()
m.degrees()




////////////////////////// Arguments //////////////////////////

// no arguments specifed
```

~/Courses/Classes/_/2022-2023/PMA/02 - Syntax/code/PMA02 - Syntax.scd   21:1   LF   UTF-8   SuperCollider

Resources

Supercollider home page

https://supercollider.github.io/

Original Supercollider home page

http://www.audiosynth.com

Code examples

http://sccode.org

Forum

https://scsynth.org

The Supercollider book

https://mitpress.mit.edu/books/supercollider-book

Eli Fieldsteel's video tutorials

https://www.youtube.com/user/elifieldsteel

Reflectives

https://www.youtube.com/channel/UCypLRZiSlIQjsT_7J4Vz35Q

Resources

A Gentle Introduction to SuperCollider - CCRMA

https://ccrma.stanford.edu/~ruviaro/texts/A_Gentle_Introduction_To_SuperCollider.pdf

Mapping and visualization with SuperCollider

http://marinoskoutsomichalis.com/mapping-and-visualization

Nick Collins tutorial

https://composerprogrammer.com/teaching/supercollider/sctutorial/tutorial.html

Thor Magnússon tutorial

http://www.ixi-software.net/content/body_backyard_tutorials.html

Stelios Manousakis course

http://modularbrains.net/portfolio/supercollider-real-time-interactive-course-sc-code/

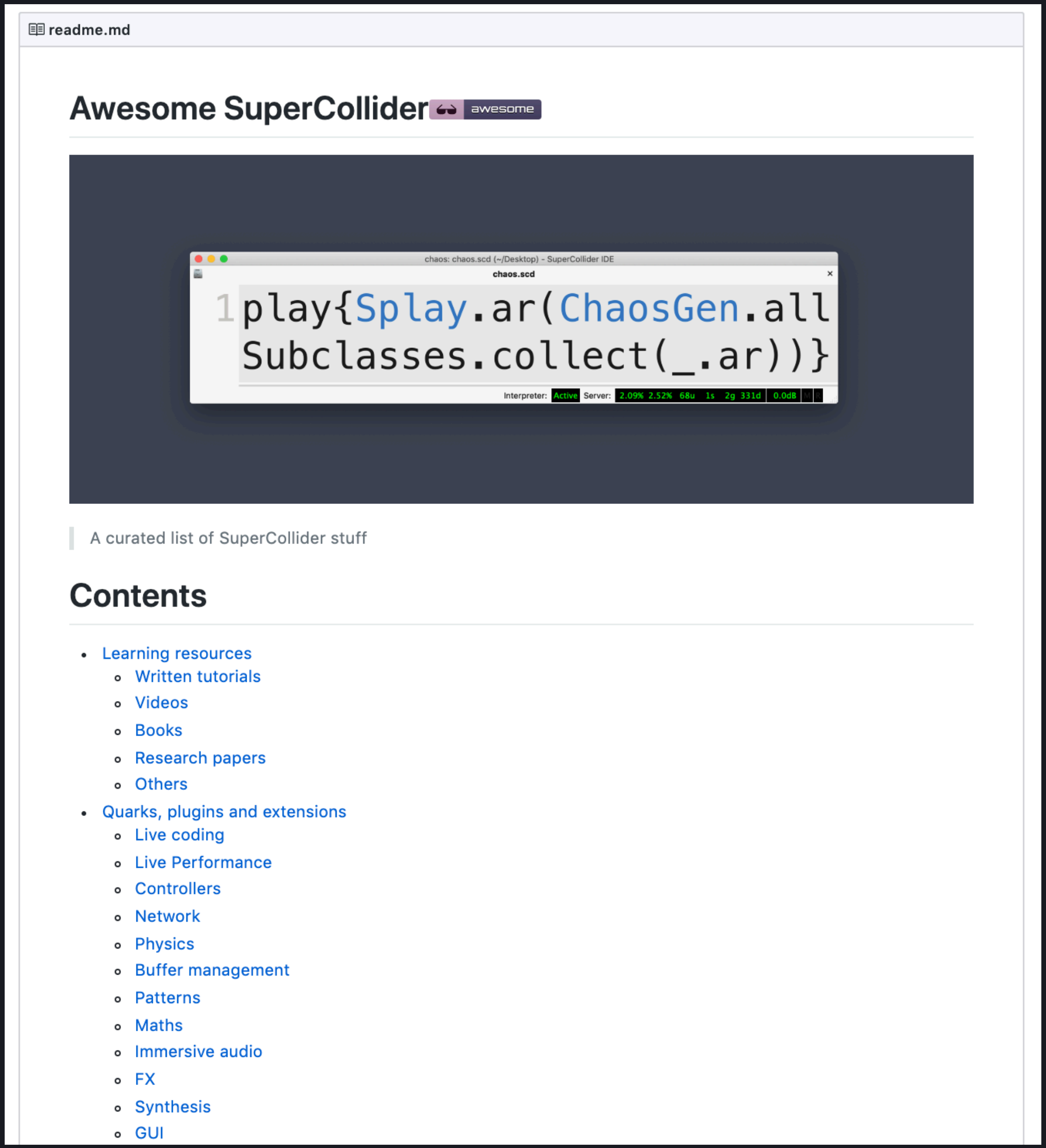
Fredrik Olofsson tutorials

http://www.fredrikolofsson.com/pages/code-sc.html

Resources

Awesome SuperCollider

https://github.com/madskjeldgaard/awesome-supercollider



readme.md

Awesome SuperCollider awesome

```
play{Splay.ar(ChaosGen.allSubclasses.collect(_.ar))}
```

> A curated list of SuperCollider stuff

Contents

- Learning resources
  - Written tutorials
  - Videos
  - Books
  - Research papers
  - Others
- Quarks, plugins and extensions
  - Live coding
  - Live Performance
  - Controllers
  - Network
  - Physics
  - Buffer management
  - Patterns
  - Maths
  - Immersive audio
  - FX
  - Synthesis
  - GUI

Basics

Objects

SuperCollider is a pure ***object-oriented programming*** (OOP) language, meaning all entities inside the program are some kind of objects.

Objects are the basic entities of the language, they bundle ***data*** and and ***methods*** that act on that data.

Two basic types of objects exist, objects with a ***fixed*** slot of data and object with a ***dynamic*** slot of data (collections).

Objects belong to a ***class*** which is a of blueprint of the object, it describes its attributes and methods.

Objects belonging to a class are called ***instances*** of that class.

Objects

A class can ***inherit*** properties and methods from another class (its ***superclass***) and then becomes its ***subclass***.

To interact with an object one sends it a ***message***.

The ***receiver*** is the one receiving the message. It looks up its own ***implementation*** corresponding to the message and then produces a ***return value*** executing that implementation.

There exist ***instance methods*** and ***class methods*** (such as new).

Instance methods are more common in SuperCollider.

Arguments

Messages to objects come with ***arguments***, these are either instances of other objects or literals.

If there are several arguments provided, they are separated by commas.

Default values for arguments can be set so that one does not have to set all arguments each time a message is passed.

The ***argument keyword*** can be used to target a specific argument in the list of all possible ones.

Objects

```
p = Point.new(1, 2)

e = Env([0,1], [1])

a = [1,2, "this is an array"]

r = Rect(2, 4, 6, 8)
r.top
r.moveTo(10, 12)

"reverse it and convert to upper".toUpper.reverse

m = Scale.major()
m.degrees()
```

Objects

```
{ SinOsc.ar }.play

{ SinOsc.ar(200, 0, 1, 0) }.play

{ SinOsc.ar(freq: 200, mul: 0.1) }.play

Array.series(10, 5, 2)

Array.series(*[10, 5, 2])

10.do({'programming'.postln})

10.do{'programming'.postln}
```

Expressions and Statements

An ***expression*** consists of values and operators for example: 1 + 2.

Expression are ***evaluated*** for making calculations and producing a ***value*** that is a result of the evaluation.

A ***statement*** is the smallest standalone element expressing an action to be carried out.

Programs are created by ***sequences*** of one or more statements.

Statements in SuperCollider are separated by ***semicolons*** ;

Expressions and Statements

```
2 * 4

"sono" ++ "logy"

["composing", "music"].choose

x = [1, 2, 3, 4].rotate(1);

if(1.0.rand >= 0.5,
{"0.5 or higher"}, {"lower than 0.5"})
```

Variables

A ***variable*** is a ***storage location*** and an ***associated identifier*** containing a value used in a program.

Variable ***names*** usually consist of letters, digits, and the underscore symbol.

Variables usually contain values or result of evaluated expressions.

Variables can be thought of as ***boxes with labels.***

Several variables can be created in one statement.

Variables must be ***declared at the beginning*** of a function.

An empty variable has the value ***nil***.

Variables

Different kind of variables exist in SuperCollider such as:

* ***Global variables*** (available everywhere)

* ***Function variables*** (available within a function)

* ***Class variables*** (available to a class)

* ***Instance variables*** (available with an instance of a class)

* ***Pseudo variables*** (provided by the compiler, *this* or *thisProcess*)

In addition to value variables, ***reference variables*** also exist which reference a variable container.

Assignments

Variables are assigned valued with ***assignment statements***.

Single assignment
The value of an expression on the right hand side is assigned to a variable on the left hand side. ***<variable> = <an expression>***

Multiple assignment
Assigns the elements of a Collection which is the result
of an expression on the right hand side, to a list of variables on the left hand side. ***<list of variables> = <expression>***

Series assignment to a list
A syntax for doing assignments to a range of values in an ArrayedCollection or List. ***<variable> = (<start>, <step> .. <end>)***

Variables and Assignments

```
Point(1, 2).x

OSCresponder.all

~myNumber = 666

this

c = 2 + 4;

# a, b, c = [1, 2, 3, 4, 5, 6];

a = (0..10);
```

Operators

An ***operator*** is a program element that is applied to one or more operands in an expression or statement.

Operators that take one operand, such as the inversion operator (neg) are referred to as ***unary*** operators.

Operators that take two operands, such as arithmetic operators (+,-,*,/), are referred to as ***binary*** operators.

Operators

Operator precedence is determined by order and parentheses.

SuperCollider supports ***operator overloading***.

Operators can thus be applied to a variety of different objects for example: ***Numbers***, ***Ugens*** and ***Collections***.

Comments

To describe code it is helpful to write ***comments*** so that when one reads it again, all explanation and detail is available and easy to grasp.

SuperCollider supports ***single*** and ***multi line*** comments.

```
// single line comment

/*
    multi
    line
    comment
*/
```

Operators and Comments

```
[1,2] ++ [3,4]

((1 + 2).asString)

1 + (2 * 2)

0.444.neg

(1.0.rand).min(0.8)

// single line comment

/*
    multi-line
    comment
*/
```

Literals

Every value in SuperCollider has a specific ***object type***.

A type determines what ***operations*** can be applied to the value.

SuperCollider is dynamically typed so variable types are usually determined during run-time.

Variables having a direct direct syntactic representation are named ***literals***.

Literals

The following literals exist:

* ***Integers***: 8, -1, 666
* ***Floats***: 0.25, -25.89
* ***Strings***, "Hello Sonology"
* ***Symbols***, 'lecture'
* ***Characters***, $a
* ***Special***, true, false, nil
* ***Literal Arrays***, #[1, 2, 'abc', "def", 4]

Boolean expressions

A boolean expression is an expression that results in a ***boolean value***, that is, in a value of either ***true*** or ***false***.

Complex boolean expressions can be built out of simple expressions, using the following boolean operators:

& (*and*, true if and only if both sides are true)

|| (*or*, true if either side is true (or if both are true))

not (*not*, changes true to false, and false to true)

Parentheses can be used for grouping the parts of complex boolean expressions.

Boolean evaluations

Arithmetic ***tests*** that can be used to create boolean values. These compare two or more objects and the ***evaluation*** returns a boolean value used for program logic.

<, less than

<=, less than or equal to

==, equal to

***!*=**, not equal to

>=, greater than or equal to

>, greater than

Conditionals

Conditional statements are used to ***test values*** and perform different actions depending on the result of the test.

The test condition must result in a ***boolean expression*** with only an option of true or false checked for in the test.

The most commonly used conditional is the ***if statement*** which tests an input and if it passes the test an action is executed.

Conditionals

The if statement usually has an ***else branch*** which specifies actions to take if the test fails.

Related conditionals are ***switch*** and ***case*** that offer many branches as well as those used for iteration on collections (while, for).

Brackets, Braces, and Parentheses

SuperCollider uses brackets, braces, and parentheses in its language syntax.

Brackets *[ ]* are used to define arrays of objects (or literals).

Braces { } are used to define function or class bodies.

Parentheses *( )* are used to express events, separate expressions or define function argument lists.

Boolean logic & Types

```
(1 == 1) || (1 == 2)

1 != 2

1 <= 2

if(0.5.coin, {"true it is"}, {"false sometimes”})

a = #["array", "that", "can't" "be", "changed"]
Array.dumpInterface

a = 'something'
b = "anything"

a.class
b.dump
a.isKindOf(Symbol)
```