

# 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

```

57
58 /* Listen to others */
59
60 { HenonN.ar(1000) * 0.2 ! 2 }.play
61
62 { GbmanL.ar(2000) * 0.2 ! 2 }.play
63
64 { StandardL.ar(3000) * 0.2 ! 2 }.play
65
66 { CuspL.ar(4000) * 0.2 ! 2 }.play
67
68 { Logistic.ar(LFNoise1.kr(0.1,0.5,3.5), LFNoise1.kr(0.2,500,1000), 0.2)
69
70 { Crackle.ar(LinCongL.ar(2).range(1.0,2.0), 0.5, 0.5) }.play;
71
72
73
74 ( // Modulate frequency
75
76 10.do {
77
78 )
79
80
81
82 ( // Fre
83
84 15.do {
85   GbmanL
86   LinC
87   * En
88   ! 2
89 )
90
91
92
93 ( // TGr
94 {
95   var
96   TGrains.ar(2,Impulse.kr(LatoocartianL.ar(150).range(5
97   LorenzL.ar(100).range(0, BufDur.kr(buf)), grSize);
98 }.play
99 )
100
101
102
103 ( // TGrains, modulate start position and rate
104 {
105   var rate = 10, buf = ~caa.bufnum;
106   TGrains.ar(2,Impulse.kr(QuadC.ar(1).range(10,150)), b
107   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:  
 lder/SuperCollider.app/Contents/Resources/HelpSource/Cle  
 link::Classes/LorenzL::  
 sc version: 3.7.0

Help browser Post window

# SuperCollider

```
26 \atk, 0.2,
27 \rel, 1.3,
28 ],
29 [ // 2
30 \ampp, Pseq([1.1,0.2,1.1,0.5], inf),
31 \durp, Pseq([1/2, 1/4, 1/2, 1/8, 1/4, 1/8, 1/12], inf),
32 \posp, Pbrown(0.005, 0.5, 0.005),
33 \atk, 0.01,
34 \rel, 1.5,
35 ],
36 [ // 3
37 \ampp, Pbrown(0.8, 1.01, 0.01),
38 \durp, Pseq([1/2, 1/2, 1/2, Pbrown(1/2, 1, 1/8, 8)], inf),
39 \posp, Pbrown(0.005, 0.5, 0.005),
40 \atk, 0.2,
41 \rel, 0.5,
42 ],
43 [ // 4
44 \ampp, Pn(Penv([0.0,0.3,0.8,0.9,0.3,0.0] * 1.3, 1 ! 5)),
45 \durp, Pseq([1/2, 1/4, 1/8, Pbrown(1/8, 1/4, 1/16, 8), 1/2, 1/2], inf),
46 \posp, Pbrown(0.0, 0.5, 0.001),
47 \atk, 0.2,
48 \rel, 0.7,
49 ],
50 [ // 5
51 \ampp, Pwhite(0.6, 1.2),
52 \durp, Pseq([1, 1/2, 1/4, Pbrown(1/16, 1/4, 1/16, 16), 1, 2], inf),
53 \posp, Pseq([1, 1/2, 1/4, Pbrown(1/16, 1/4, 1/16, 16), 1, 2], inf),
54 \atk, 0.4,
55 \rel, 0.6,
56 ],
57 [ // 6
58 \ampp, Pn(Penv([0.1, 1.1, 0.5, 0.1], [4, 4, 8])),
59 \durp, Pseq([1/8, Pbrown(1/32, 1/8, 1/16, 32), 1/6, 1/3], inf),
60 \posp, Pbrown(0.005, 0.5, 0.01),
61 \atk, 0.1,
62 \rel, 0.3,
63 ],
64 [ // 7
65 \ampp, Pbrown(0.9, 1.3, 0.01),
66 \durp, Pseq([
67   Pseq([1, 1, 1/2, Pbrown(1/32, 1/4, 1/8, 32)], 4),
68   Pseq([1, 1/2, 1, Pbrown(1/8, 1/4, 1/32, 32)], 4)
69 ], inf),
70 \posp, Pkey(\durp) * 0.5,
71 \atk, 0.1,
72 \rel, 0.5,
73 ],
74 ];
75
```



The image shows the SuperCollider software interface. On the left is a code editor with a SynthDef definition. In the center-right is a 'Soundfile - Looper' window displaying two audio waveforms. Below that is a 'SynthDef - Designer' window with controls for frequency (4010) and amplitude (0.5), and buttons for 'Play Note', 'Prepare Rec.', 'Randomize', 'Print', and 'Play Pattern'. At the bottom right, there's a status bar showing 'Interpreter: Active' and 'Server: 7.26% 10.33% 1093u 182s 2g 181d 0.0dB'.

Soundfile - Looper

Colorful

SynthDef - Designer

freq 4010

amp 0.5

Play Note Prepare Rec. Randomize Print

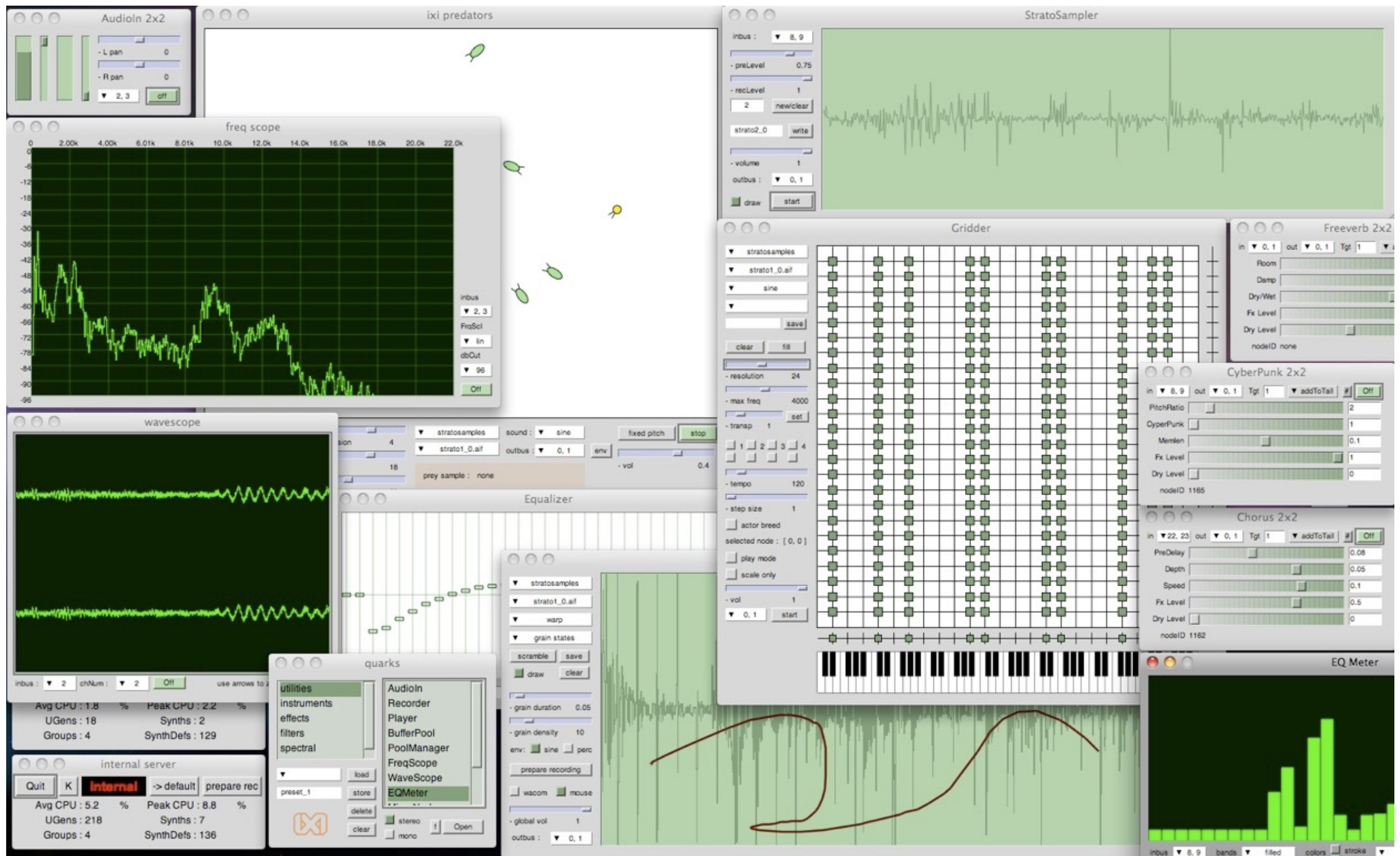
First Play Pattern

Help browser Post window

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



# 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. slang 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 slang 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.

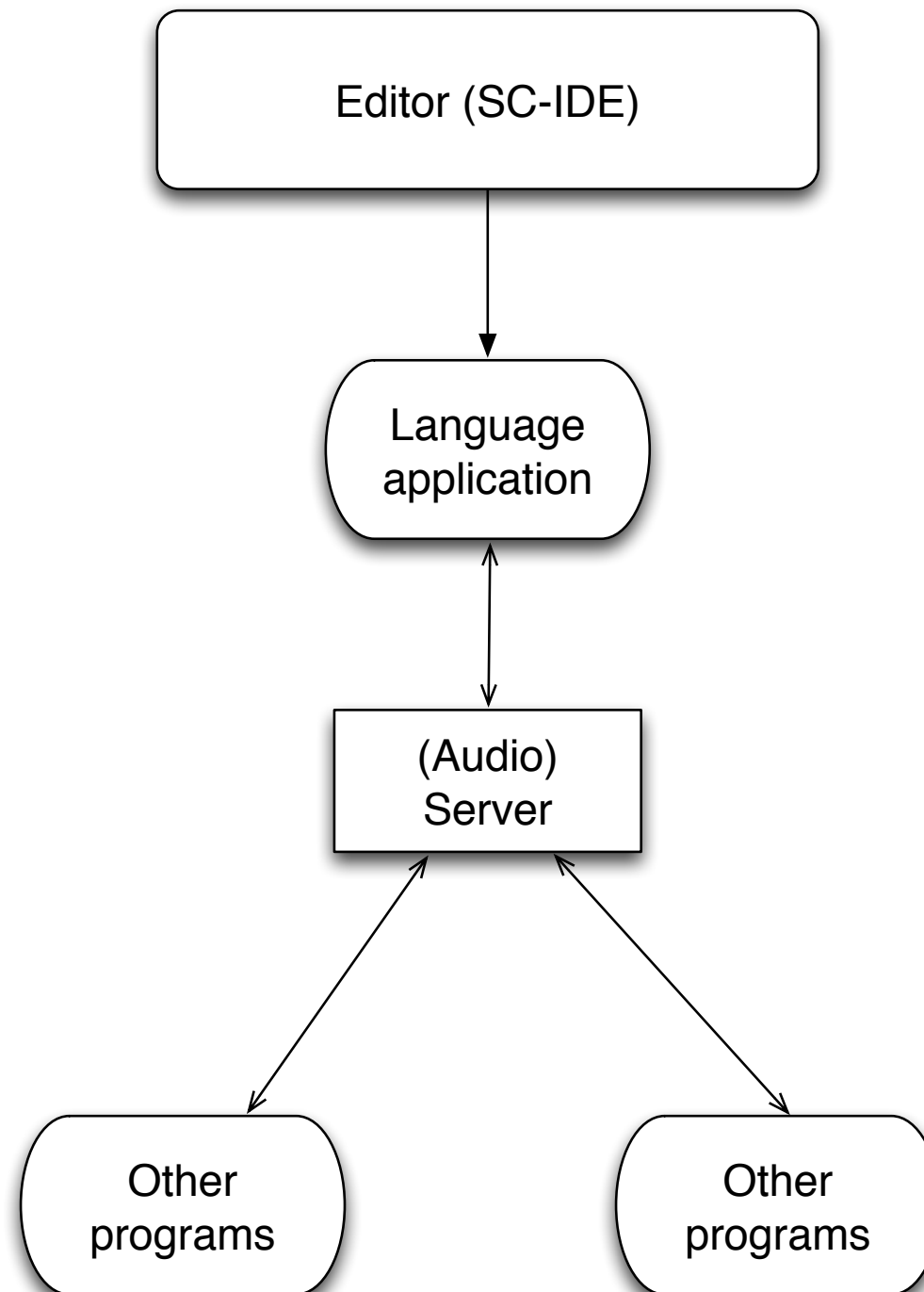
*Audio channels* that SuperCollider synths use for sending their sound through are called audio **buses**.

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

# Architecture

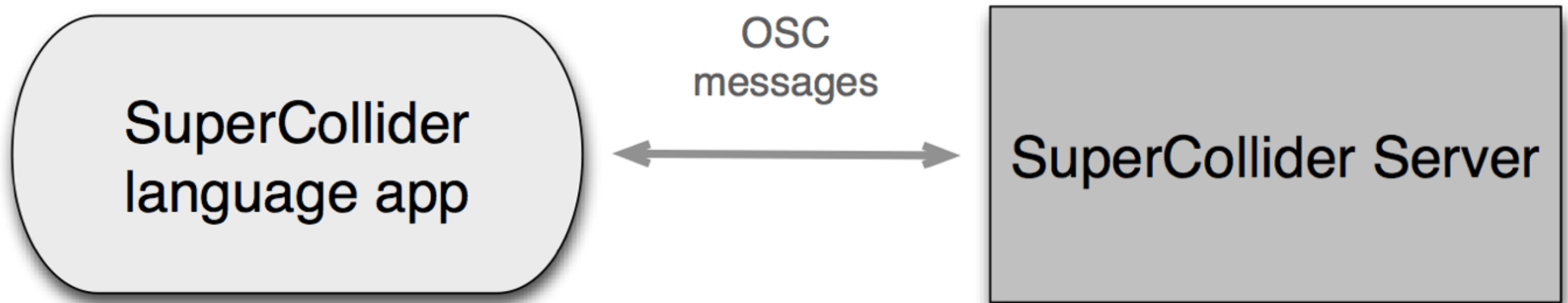
---

SuperCollider  
supercollider.sourceforge.net  
Version 3.6



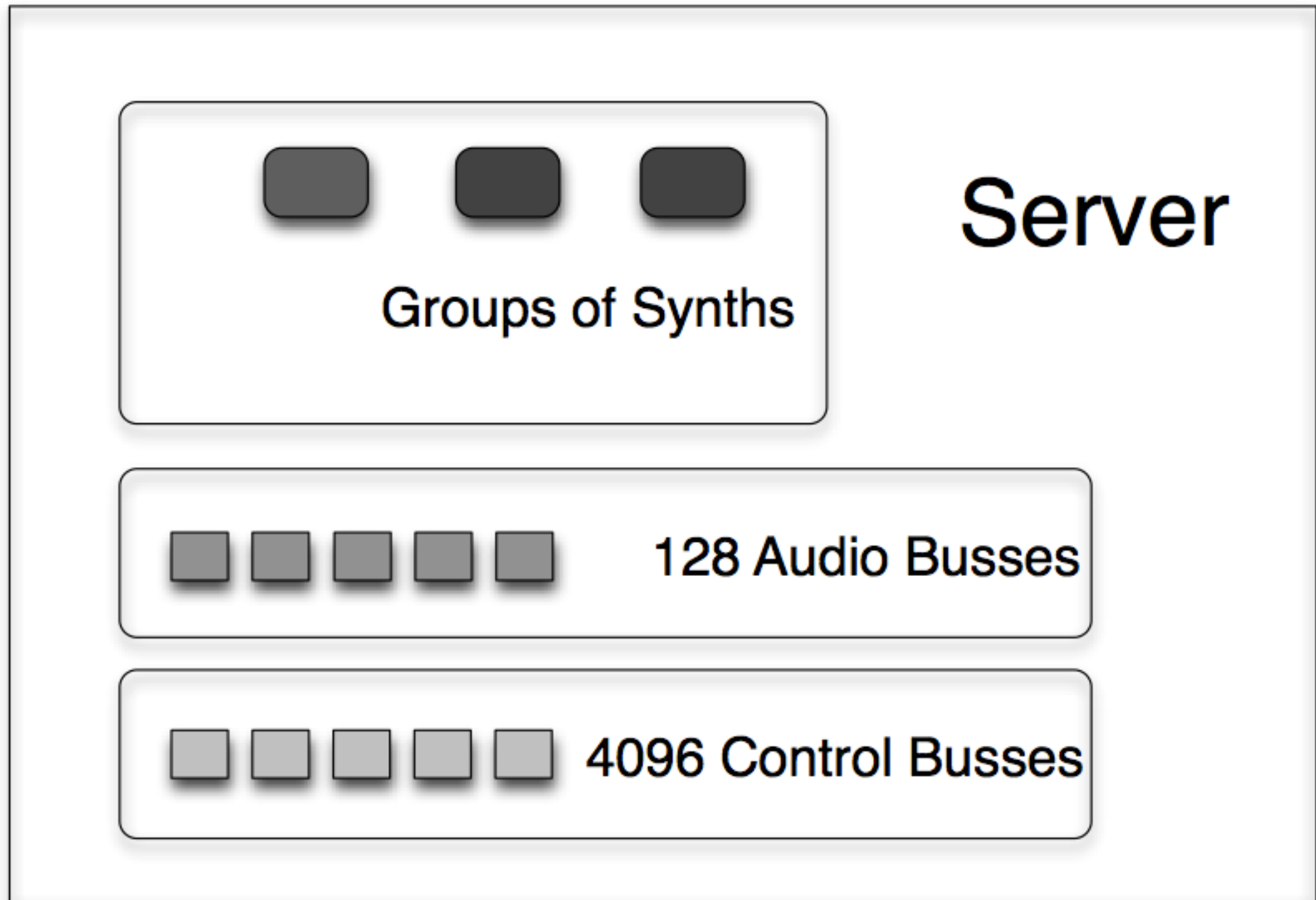
# *Messaging*

---

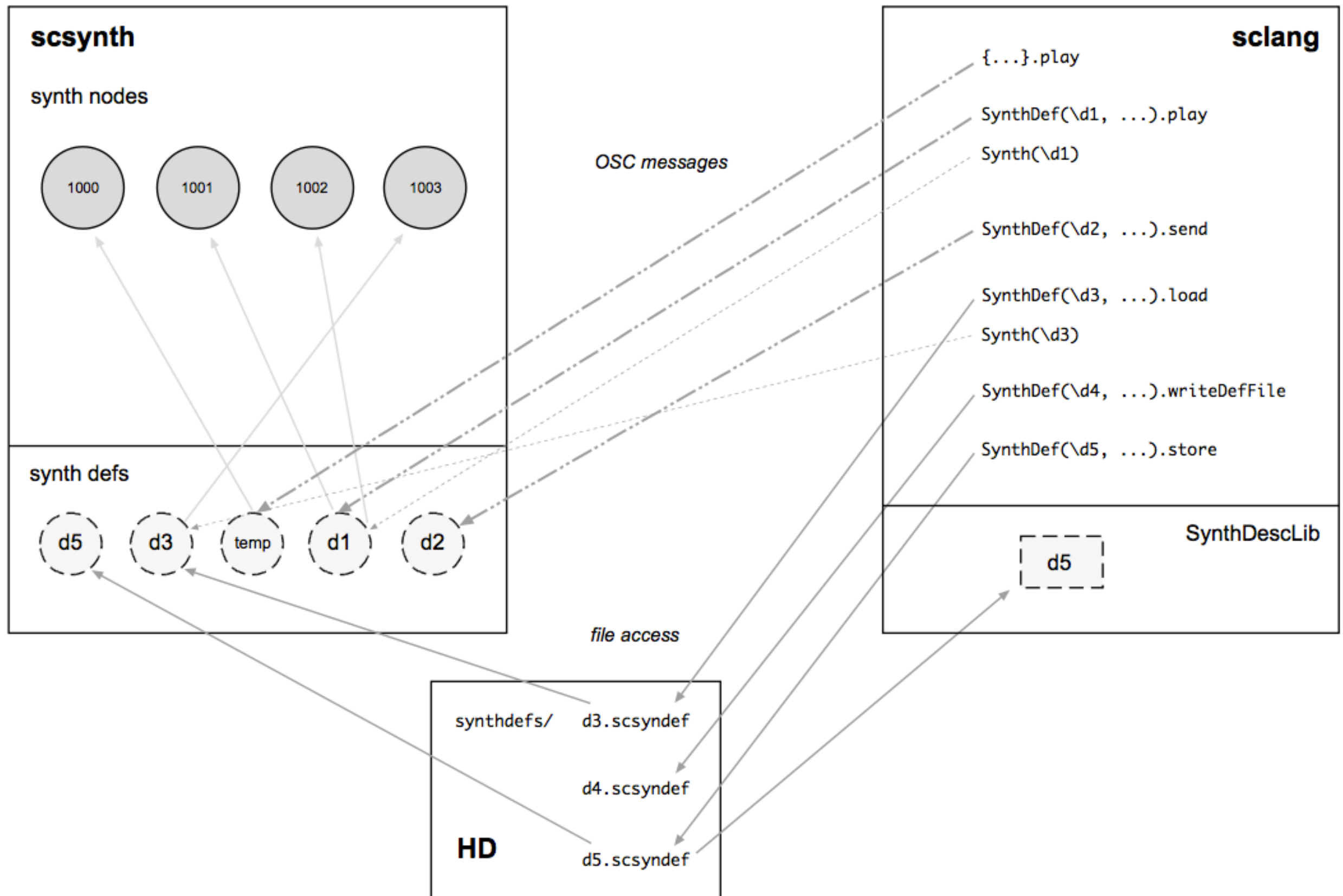


# Architecture

---

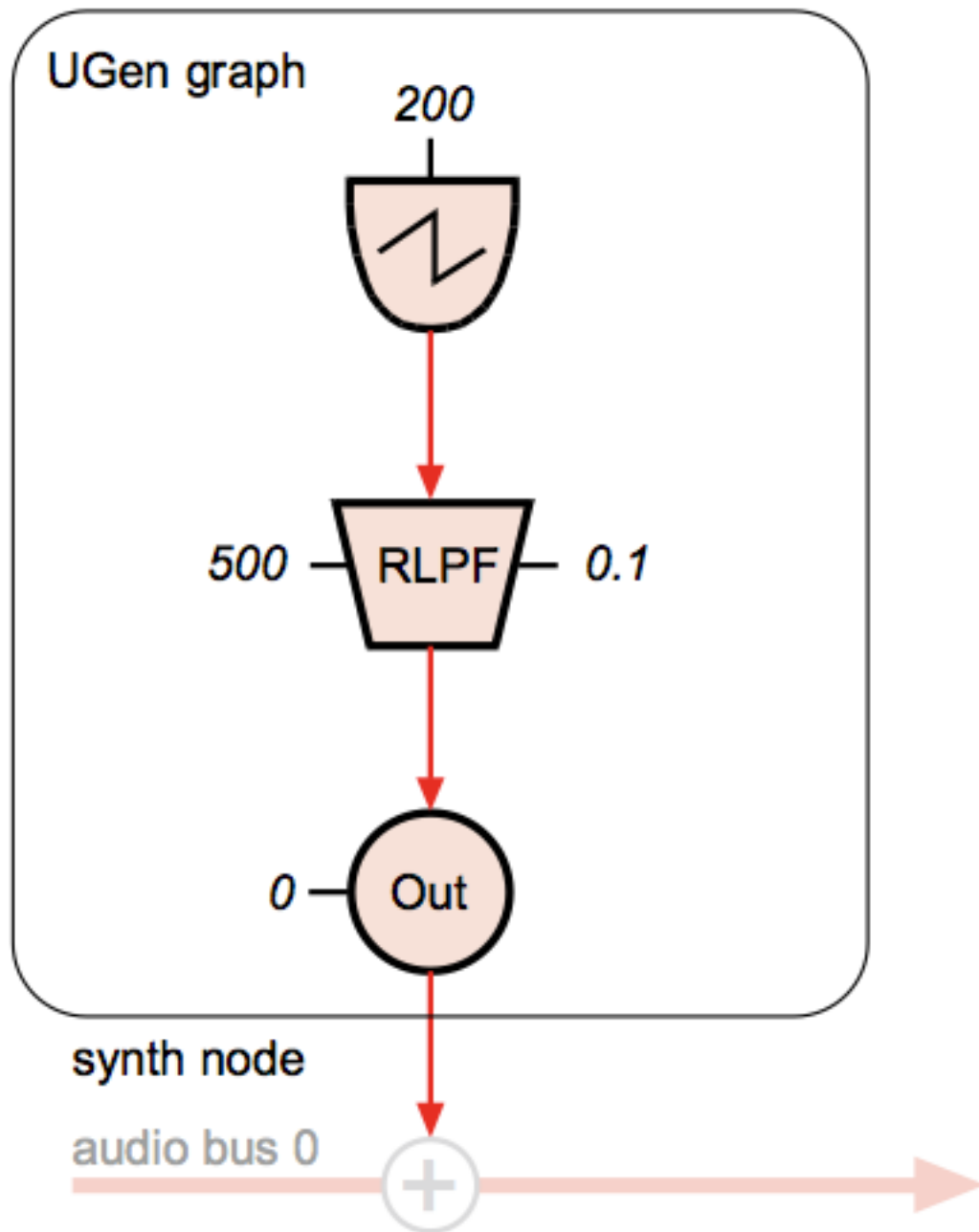


# Architecture





# 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

PhysicalModeling.scd

```
24
25 scope
26 // 1. NOISES/SYNTHESIS
27 ~eObjects1 = List.new;
28 ~eObjects1.add(BNetworkItem.new(BLSynth1, [\outBus, 0, \curveType, 3, \duration, ~globalDur],
  "Waveth"));
29 ~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"));
30 ~eObjects1.add(BNetworkItem.new(BLSynth7, [\outBus, 0, \curveType, 4, \duration, ~globalDur],
  "Pulcao"));
31 ~eObjects1.add(BNetworkItem.new(BLSynth3, [\outBus, 0, \duration, ~globalDur], "Gritera"));
32 ~eObjects1.add(BNetworkItem.new(BLSynth4, [\outBus, 0, \duration, ~globalDur], "Trainir"));
33 ~eObjects1.add(BNetworkItem.new(BLSynth2, [\outBus, 0, \osc, 1, \curveType, 4, \duration,
  ~globalDur], "Hecton"));
34 ~eObjects1.add(BNetworkItem.new(BLSynth6, [\outBus, 0, \osc, 0, \curveType, 3, \duration,
  ~globalDur], "Narrov"));
35 ~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, ~globalDur], "Litari"));
```

Plot

Help browser

Home Browse Search Indexes Randomness - Table of contents

## 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>

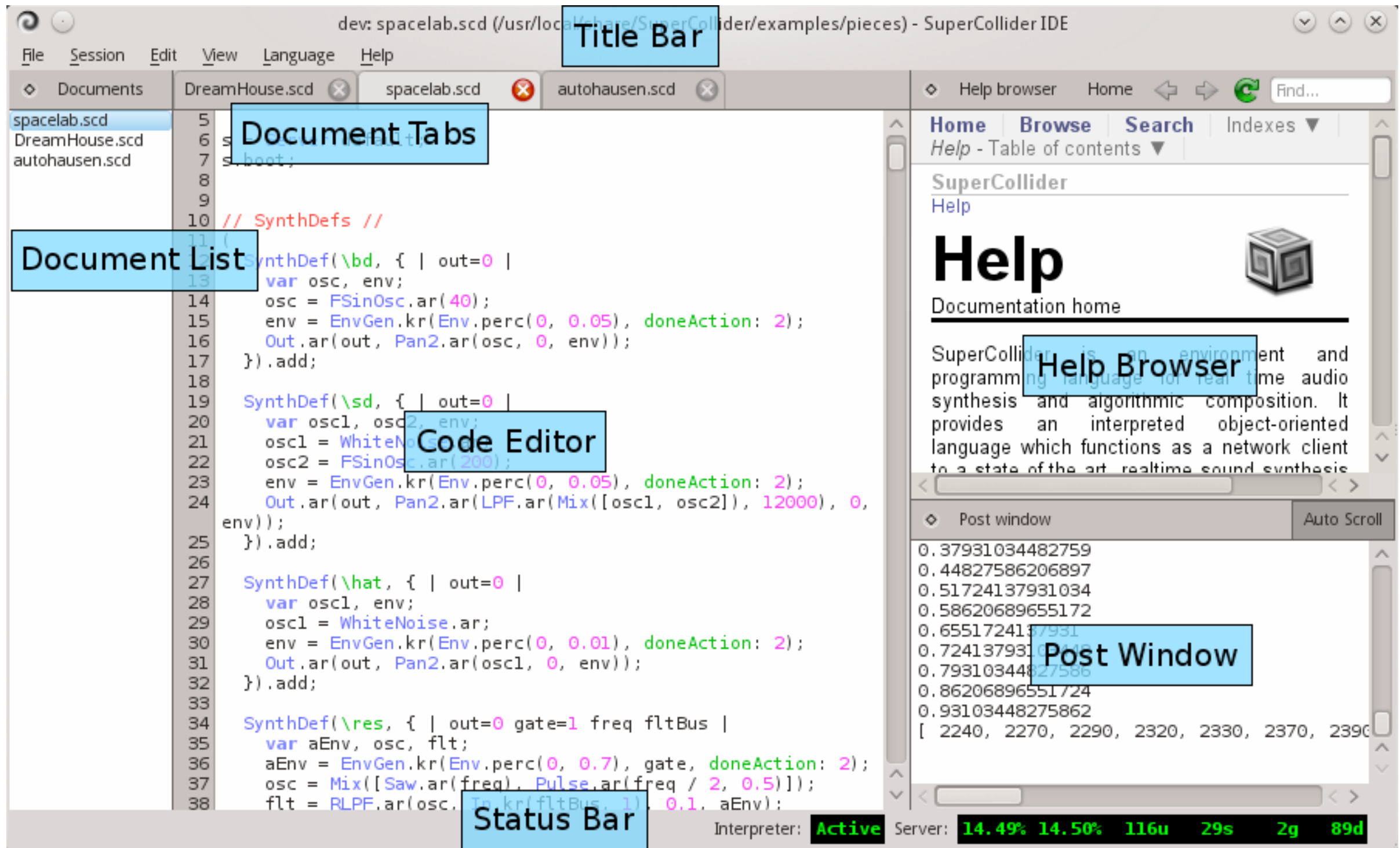
```
5.rand // evenly distributed.
1.0.linrand // probability decreases linearly from 0 to <number>.
```

Between <number> and <number>

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 init'd in 0.06 seconds  
RESULT = 0  
Welcome to SuperCollider 3.6.6. For help press Cmd-D.

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

# IDE



# 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 specified
```

~/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/UCypLRZiSIIQjsT\\_7J4Vz35Q](https://www.youtube.com/channel/UCypLRZiSIIQjsT_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](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**

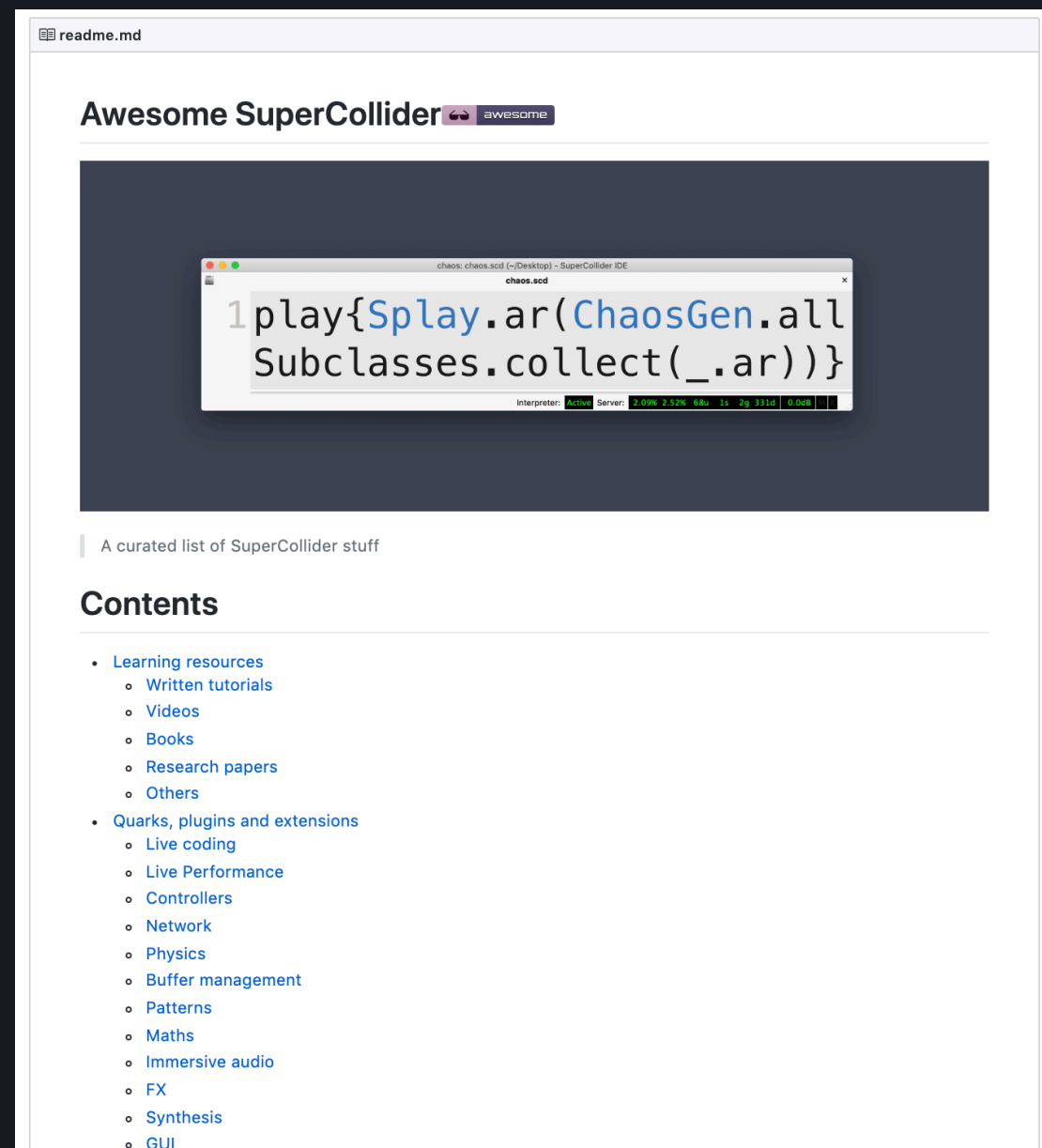
*<http://www.fredrikolofsson.com/pages/code-sc.html>*



# Resources

## Awesome SuperCollider

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



*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 **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 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".toUpperCase.reverse
```

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

```
m.degrees()
```



# Arguments

---

```
{ 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

OSResponder.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 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)
```

