Workshop notes for SuperCollider pattern introduction workshop september 2020

2020-09-01T15:32:13+02:00

Contents

These are workshop notes for the online introductory SuperCollider workshop put on by Notam in September 2020. It covers the basics of using the pattern library in SuperCollider

You can download the notes here: workshop-notes

Design

Short history of SuperCollider

SC was designed by James McCartney as closed source proprietary software

Version 1 came out in 1996 based on a Max object called Pyrite. Cost 250$+shipping and could only run on PowerMacs.

Became free open source software in 2002 and is now cross platform and is now maintained by a wonderful group of developers.

Overview

When you download SuperCollider, you get an application that consists of 3 separate programs:

1. The IDE, a smart text editor
2. The SuperCollider language / client (**sclang**)
3. The SuperCollider sound server (**scsynth**)

Architecture

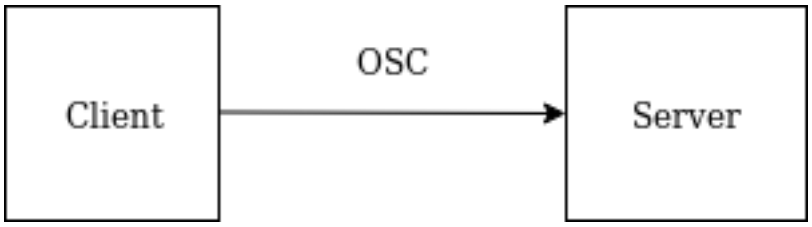

Figure 1: alt

The client (language and interpreter) communicates with the server (signal processing)

This happens over the network using Open Sound Control

Multiple servers

This modular / networked design means one client can control many servers

---

Consequences of SC's modular design

Each of SuperCollider's components are replacable:

IDE <—> SCIDE, (N)Vim, Atom or VSCode

language <—> Python, CLisp, TidalCycles, Javascript

server <—> Max/MSP, Ableton Live, Reaper

Extending SuperCollider

The functionality of SuperCollider can be extended using external packages

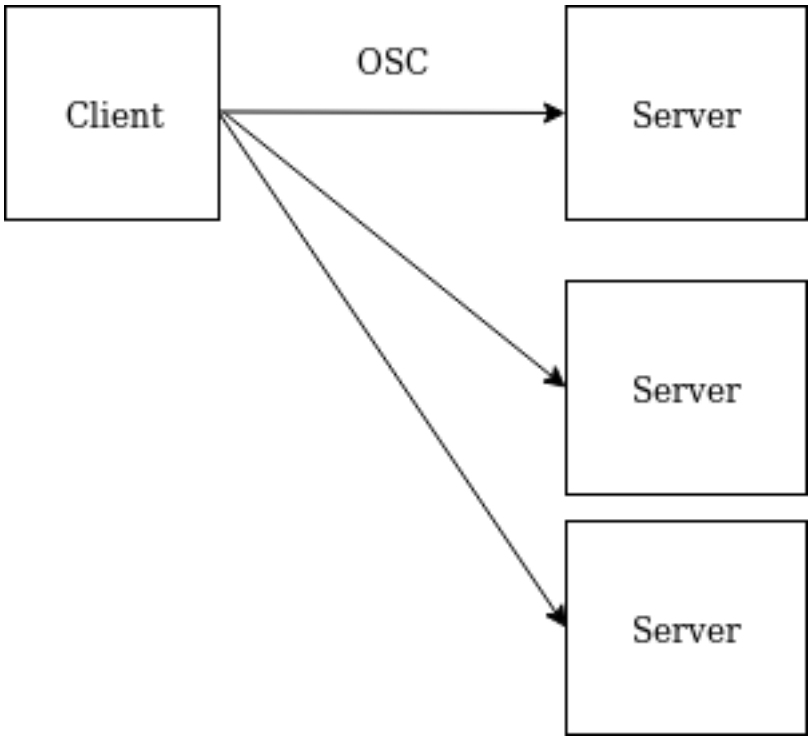

Figure 2: alt

These are called Quarks and can be installed using SuperCollider itself

```
// Install packages via GUI (does not contain all packages)
Quarks.gui;

// Install package outside of gui using URL
Quarks.install("https://github.com/madskjeldgaard/KloudGen");
```

SC Plugins

SC3 Plugins is a collection of user contributed code, mostly for making sound

The plugins are quite essential (and of varying quality / maintenance)

How to use the IDE

The IDE is the text editor that comes with SuperCollider. It has some really smart features that are really helpful when writing code.

Important keyboard shortcuts

- Open help file for thing under cursor: **Ctrl/cmd + d**
- Evaluate code block: **Ctrl/cmd + enter**
- Stop all running code: **Ctrl/cmd + .**
- Start audio server: **Ctrl/cmd + b**
- Recompile: **Ctrl/cmd + shift + l**
- Clear post window: **Ctrl/cmd + shift + p**

The IDE as a calculator

SuperCollider is an interpreted language

This means we can "live code" it without waiting for it to compile

A good example of this is using it as a calculator. Try typing `2+2` and evaluate it:

```
2+2
-> 4
```

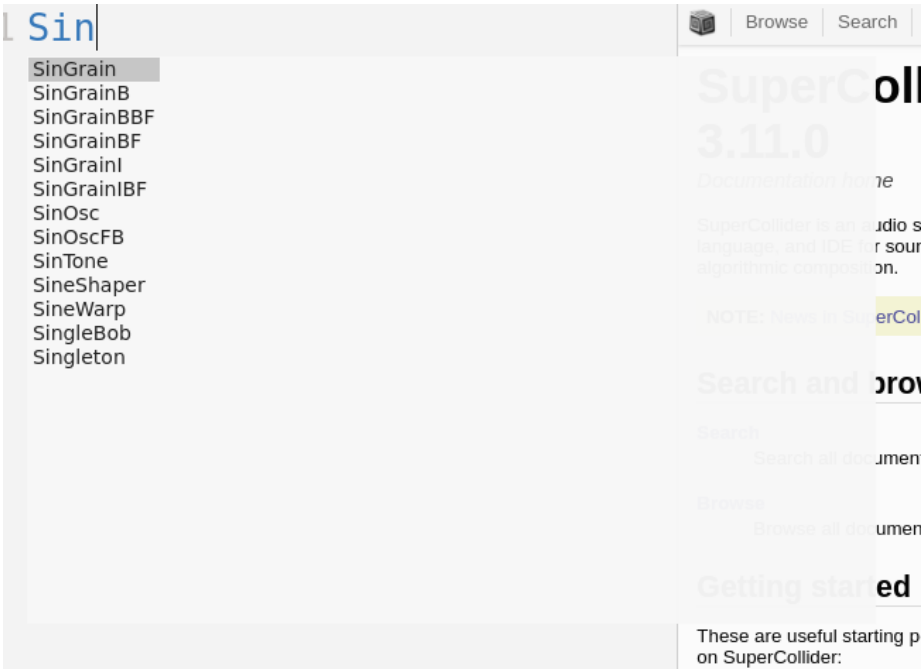

Figure 3: autocomplete

Autocompletion

Start typing `Sin` and see a menu pop up with suggestions (and help files).

Use `up/down` arrow keys to navigate and hit `enter` to choose one

The status line

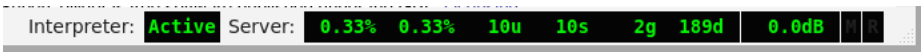

Figure 4: autocomplete

Shows information about system usage

Right click to see server options + volume slider

Help browser

There is an interactive help browser available.

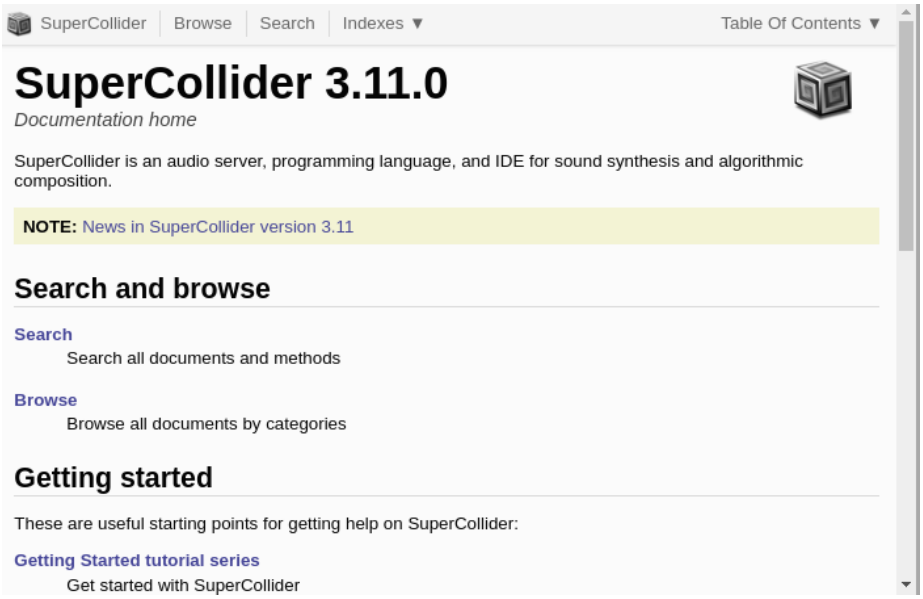

Figure 5: autocomplete

You can select and evaluate all code in the browser and see / hear the results immediately.

Help browser online

There's an online version of the help system available at doc.sccode.org/ which is really helpful for sharing links to documentation.

Post window

This is where you see the resulting return messages of the code you have evaluated

This is also where you see error messages posted.

About patterns

What is a pattern?

From the Pattern help file:

> "[The Pattern] classes form a rich and concise score language for music"

In other words:

Patterns are used to sequence and compose music

The cool thing about this is it is music treated as data.

This means we can easily transpose, stretch and warp the composition like you will see in the following (instead of manually doing all of these things in a DAW).

This means for example that composing a 4 bar loop is not necessarily any more or less work than a 4 hour one.

Guides in the help system

Patterns are pretty well documented in the help system:

- A practical guide
- Understanding Streams, Events and Patterns

Event patterns - Sequence sound events

What is a (sound) event?

Think of what happens when you press the key of a piano

What data does that involve?

- Duration of key press
- Pitch of the key
- Sustain (are you holding the foot pedal?)
- etc. etc.

What does an Event look like?

Make sure your server is booted before trying this:

```
// See the post window when evaluating these
().play; // Default event
(freq:999).play;
(freq:123, sustain: 8).play;
```

Changing the default synth

The default synth sucks

You can change it by defining a new synth called `\default`

More info on my website

Introducing the almighty Pbind

Arguably the most important pattern class in SuperCollider

A Pbind simply consists of a list of key/value pairs.

In this example, the keys are on the left side (`\dur` and `\degree`) and the values on the right (`1/4` and `0`).

```
// Play quarter notes (try changing the 0 to another integer)
Pbind(
    \dur, 1/4,
    \degree, 0
).play
```

Note the commas: This is what makes it a list.

Keys correspond to Synth arguments

Most often, keys correspond to a Synth's arguments.

Example: If a SynthDef has the argument cutoff, we can access that argument in a Pbind using `\cutoff`.

Important note on the dur key

`\dur`is used in most SynthDefs to specify the duration of a note/event.

Make sure this key never gets the value 0.

When does a Pbind end?

If one of the keys of a Pbind are supplied with a fixed length value pattern, the one running out of values first, will make the Pbind end.

Live coding: Pdef

Live coding patterns: Wrap your event pattern (Pbind) in a Pdef:

```
Pdef('myCoolPattern', Pbind(...)).play;
```

The Pdef has a name 'myCoolPattern' which is a kind of data slot accessible throughout your system

Every time you evaluate this code, it overwrites that data slot (maintaining only one copy)

Value patterns - Generate data for your event patterns

The building blocks of compositions

Basic building blocks:

- List patterns - `Pseq`
- Random value patterns, eg `Pwhite`, `Pbrown`
- Random sequence patterns - `Pshuf`, `Prand`, etc.
- Rests

Slightly more advanced building blocks:

- Envelope patterns like `Pseg` or `Pstep`
- Data sharing between event parameters, eg `Plambda` -
- Patterns in patterns

Advanced:

- Generate patterns / Pattern spawning - eg. `Pspawn`

List patterns

See all of them here

Testing value patterns: asStream

You will see the .asStream method a lot in the documentation for value patterns.

```
// Pattern
p  Pseq([1,2,3]);

// Convert to stream
```

```
p  p.asStream;

// See what values the pattern produces
p.next; // 1, 2, 3, nil
```

Pseq: Classic sequencer

```
// Play values 1 then 2 then 3
Pseq([1,2,3]);

// 4 to the floor
Pseq([1,1,1,1]);
```

Random value patterns: Pwhite and Pbrown

```
// (Pseudo) random values
Pwhite(lo: 0.0, hi: 1.0, length: inf);

// Drunk walk
Pbrown(lo: 0.0, hi: 1.0, step: 0.125, length: inf);
```

Random sequence patterns: Prand and Pxrand

```
// Randomly choose from a list
Prand([1,2,3],inf);

// Randomly choose from a list (no repeating elements)
Pxrand([1,2,3],inf);
```

Probability: Pwrand

Choose items in a list depending on probability

```
// 50/50 chance of either 1 or 10
Pwrand([1, 10], [0.5, 0.5])

// 25% chance of 1, 25% change of 3, 50% chance of 7
Pwrand([1, 3, 7], [0.25, 0.25, 0.5])

// 30% chance of 3, 40% change of 2, 30% chance of 5
Pwrand([4, 2, 5], [0.3, 0.4, 0.3])
```

Envelope pattern: Pseg

```
// Linear envelope from 1 to 5 in 4 beats
Pseg( levels: [1, 5], durs: 4, curves: \linear);

// Exponential envelope from 10 to 10000 in 8 beats
Pseg( levels: [10, 10000], durs: 8, curves: \exp);
```

Rest

Skip/sleep a pattern using Rest. If used in the `\dur`key of a Pbind, the value in the parenthesis is the sleep time

```
// One beat, two beats, rest 1 beat, 3 beats
Pbind(\dur, Pseq([1,2,Rest(1),3])).play;
```

Pkey: Share data between event keys

Using Pkey we can make an event's parameters interact with eachother

```
// The higher the scale degree
// ... the shorter the sound
Pbind(
    \degree, Pwhite(1,10),
    \dur, 1 / Pkey(\degree)
).play
```

More info about data sharing in patterns: here

---

patterns in patterns: The computer music inception

You can put patterns in almost all parts of patterns.

This may lead to interesting results:

```
// A sequence with 3 random values at the end
Pseq([1,2,Pwhite(1,10,3)]);

// An exponential envelope of random length
Pseg(levels: [10, 10000], durs: Pwhite(1,10), curves: \exp);
```

Working with pitches

The pattern library comes with a bunch of useful features for working with pitches in a convenient and interesting way.

The pitch model

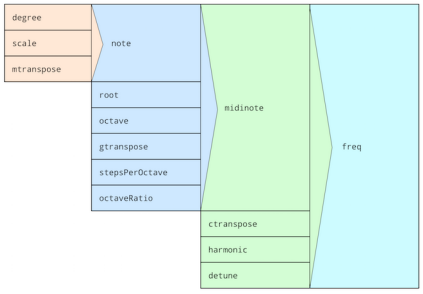

Figure 6: pitch model

Pitch model is described here

Changing scales

```
// Use the \scale key, pass in a Scale object
Pbind(\scale, Scale.minor, \degree, Pseq((1..10))).play;
Pbind(\scale, Scale.major, \degree, Pseq((1..10))).play;
Pbind(\scale, Scale.bhairav, \degree, Pseq((1..10))).play;
```

Available scales

```
// See all available scales
Scale.directory.postln
```

Changing root note

```
// Use the \root key to transpose root note (halftones)
```

```
Pbind(\root, 0, \degree, Pseq((1..10))).play;
Pbind(\root, 1, \degree, Pseq((1..10))).play;
Pbind(\root, 2, \degree, Pseq((1..10))).play;
```

Changing octaves

```
// Use the \octave key
Pbind(\octave, Pseq([2,4,5],inf), \degree, Pseq((1..10))).play;
Pbind(\octave, Pwhite(3,6), \degree, Pseq((1..10))).play;
Pbind(\octave, 7, \degree, Pseq((1..10))).play;
```

Playing chords

```
// Add an array of numbers to the degree parameter
// to play several synths at the same time (as a chord)
Pbind(\degree, [0,2,5] + Pseq([2,4,5],inf), \dur, 0.25).play;
```

Changing tempo

The tempo of patterns are controlled by the TempoClock class You can either create your own TempoClock or modify the default clock like below

```
TempoClock.default.tempo_(0.5) // Half tempo
TempoClock.default.tempo_(0.25) // quarter tempo
TempoClock.default.tempo_(1) // normal tempo
```

Further learning resources

Videos

Tutorials by Eli Fieldsteel covering a range of subjects: SuperCollider Tutorials

---

E-book

- A gentle introduction to SuperCollider

Paper:

- Introduction to SuperCollider, Andrea Valle

- The SuperCollider Book

Community

- scsynth.org
- sccode.org
- Slack
- Lurk
- Mailing list
- Telegram
- Telegram ES
- Facebook

Awesome SuperCollider

A curated list of SuperCollider stuff

Find inspiration and (a lot more) more resources here:

Awesome Supercollider

Learning to code: Advice

- Practice 5 minutes every day
- Set yourself goals: Make (small) projects
- Use the community
- Contribute to SuperCollider - improve documentation, help out on the forums or make bug reports

---