Appendix: Syntax of the SuperCollider Language

Iannis Zannos

The following is not a formal exposition of the syntax rules but a summary to help the reader understand the code of the examples in this book.

A.1 Comments

Comments are written as in C++, Java, PHP, or similar languages

- Multiline comments are enclosed between `/*` and `*/`
- Single-line comments start at // and run to the end of the line

A.2 Identifiers

Identifiers are sequences of alphanumeric characters and the underscore character _ that do not start with a capital letter. Such a sequence may be one of the following:

- The name of a variable or argument. Variables are declared in functions, methods, or classes, and arguments in functions or methods. Example: `{arg freq; /* function code ... */}`
- The name of a message. Message names must correspond to method names
- The variable and argument declaration keywords `arg` and `var`
- The special keywords `this` and `super`
- The constants `pi`, `inf`, `nan`, `true`, `false`

A.3 Literals

Literals are objects whose value is represented directly in the code (rather than computed as a result of sending a message to an object). Literals in SuperCollider are

- Integers (e.g., `–10`, `0`, `123`) and floating-point numbers (e.g., `–0.1`, `0.0`, `123.4567`), which can be in exponential notation (e.g., `1e–4`, `1.2e4`), alternate radices up to base 36 (e.g., `2r01101011`, `12r4A.A`), or combined with the constant `pi` (e.g., `2pi`, `–0.13pi`)

- Strings, enclosed in double quotes: `"a string"`
- Symbols, enclosed in single quotes (`'a symbol'`) or preceded by `\` (`\a_Symbol`)
- Literal arrays: Immutable arrays declared by prepending the number sign (`#`)
- Classes: A Class is represented by its name. Class names are like identifiers but start with a capital letter
- Characters (instances of Char), a single character preceded by the dollar sign `$` (e.g., `$A`, `$a`). Nonprinting characters (tab, linefeed, carriage return) and backslash are preceded by a backlash (e.g., `$\n`, `$\t`, `$\\`)
- Identifiers (as described in section A.2 above); see also the SuperCollider Help file on Literals)

A.4 Primitives

Primitives appear in the code of certain methods in class definitions and are identifiers that start with an underscore _. They call code that is compiled in C++ and perform elementary operations of the language, which cannot be coded in SuperCollider

A.5 Grouping Elements

Parentheses () are used to:

- Group expressions in order to specify order of execution: `1 + (2 * 3)`
- Enclose arguments that accompany messages: `2.pow(3)`
- Create numerical arrays from "Matlab type" series notation: `(1..5)`
- Create Events from keyword-value pairs: `(freq: 440, amp: 0.1)`

Brackets [] are used to:

- Create Arrays or other types of collections `[1, 2, 5]`
- Index into collections for reading or writing of values: `aDictionary[\freq]`,
- `anArray[0]`

Braces {} are used to:

- Define functions: `{arg a, b; a + b}`
- Define Classes: `Nil {/* class definition code */}`
- Define methods: `isNil {^true}`

A.6 Binary Operators

Many arithmetic, logical, stream, and other binary operator symbols are used much as they are in C++.

A.7 Delimiters

• The dot . is used in the following senses
• To attach a message to the receiving object that it is sent to: `123.squared`
• To append an *adverb* to a binary operator. (Adverbs are identifiers or Integers that modify the behavior of an operator): `(1..15) *.f [1, 10, 100]`
• Triple dots . . . are used to collect multiple arguments into an array, in argument definitions: `{ |... args|` or in multiple variable assignments
• `#a, b ... rest = [1, pi, 10, true, inf];`
• Double dots are used in notation of arithmetic series: `(1..5)`, `(0, 0.1 .. 10)`
• Comma is used to separate arguments `f.value(pi, 400)`, elements of Collections `[pi, \a, 5]`, or variables or arguments in declaration statements `{arg a, b;` or in multiple assignment statements `#freq, amp = #[440, 0.1]`
• Semicolon is used to separate statements. The last statement of a program (function) does not need to end in a semicolon. `a.postln; b = a.squared`
• Pipe signs | | are used to delimit an argument declaration statement `{| a, b | a + b}`. This is alternative notation to `arg a, b;`

A.8 Special Characters

• ^ marks the statement that it precedes as a return value statement in a method: `^a * 2`
• * Has 2 uses:
• Preceding an argument in a message, it applies the collection's elements as separate arguments: `foo.value(*[1, 2, 10.rand]);`
• Preceding the name of a method in Class definition code, it indicates that this is a class method: `*initClass {all = IdentityDictionary.new}`
• # (number sign) is used as prefix in 2 cases:
• Multiple variable assignment: `#freq, amp = [400, 0.1]`
• Construction of immutable Arrays and closed Functions: `#[1, 5, 11]`, `#{pi ! 3}`
• $ precedes Character instances: `$a $. $A`
• ~ Tilde before an identifier treats it as an environment variable (see chapter 5, section 5.3.5: Environment Variables)
• < and > construct accessor methods for variables in classes: `var <>freq`

A.9 Construction of Specific Kinds of Objects, Abbreviations, Various Conventions

In addition to those covered above (`"string," 'symbol,' \symbol`) there are the following constructor elements:

- Braces {} construct Functions: `{|a, b| a * b}`
- Brackets [] construct Arrays (or other Collections when preceded by collection name): `[1, \a]` `Set[1, a]`, `Dictionary[\a->1, \b->2]`
- Parentheses () enclosing keyword-value pairs construct events: `(a: 1, b: 2)`
- "At" sign @ between 2 numbers constructs a Point: `-5@10`
- "At" sign @ between a collection and a number indexes into the collection: `[1, 5, 7]@1`
- Arrow -> between values constructs an association: `\freq->440`
- Underscore _ by itself in a message statement constructs a function: `_.isPrime` (see the "Partial Application" Help file)
- `element ! n` duplicates the element n times (evaluating it if it's a Function) and collects the result in an Array: `{10.0.rand} ! 12`
- The message `new` can be omitted between a Class name and arguments enclosed in parentheses: `Synth("sine")` is equivalent to `Synth.new("sine")`
- The message `value` can be omitted between a Function and arguments enclosed in parentheses: `foo.(n)` is equivalent to `foo.value(n)`. Also, `someFunc.()` is equivalent to `someFunc.value`.
- Functions as sole arguments need not be enclosed in parentheses: `10.do {"hello" .postln}`
- Messages whose name ends in underscore _ can also be written in "variable assignment" format: `aPoint.x_(0)` is equivalent to `aPoint.x = 0`
- < or > prepended to a variable name in a variable declaration statement in a Class constructs corresponding methods for getting or setting the value of that variable. `var <>x, <>y;`