# SoundString analysis

The execution is performed in ticks, each tick is the smallest time unit possible and represents small timeframe. The Soundstring works by starting with a single play thread, which goes from symbol to symbol and generates audio output, this thread can fork into several others.

Each thread has properties, like volume, freuqncy and wave shape. These can be changed by placing a special symbol. Sound is generated every time a symbol representing frequency is encountered. In every tick, the interpreter reads all the characters until it finds another one that represents frequency, at which it stops and generates sound with appropriate duration, that’s currently set for given thread.

Symbols:

0 – 9 = duration of the note  
a – Z = play tone of given frequency  
. = pause  
~ = set sine waveform  
/ = set sawtooth waveform  
| = set pluck waveform  
- = set square waveform  
^ = set triangle waveform  
\* = set noise waveform  
( … ) = acord – group several statements that are played at once. It’s considered as one note, even if it contains several  
{name; sequence} = subroutine definition  
{name: parameters} = subroutine call, parameters are optional  
$n = argument usage in subroutine, n is number from 0 to 9  
[name: sequence :n] = repeat sequence n times, 0 for infinity, name is optional  
[name] = stop sequence with given name, if playing  
&n = perform shift of pitch scale  
&+n = perform shift of pitch scale plus 1 more step  
&-n = perform shift of pitch scale minus 1 more step  
@bpm; = set bpm  
#soundfont; = specifies a file with alternate soundfont that will be used  
> = set the tones to use fadeout  
< = set the tones to use fadein  
= = set the tones to use constant volume  
%n = set the volume, n is an alphabet letter  
?n = skip over following statement if argument n is zero  
!n = skip over following statement if argument n is nonzero  
?(n=x) = skip over following statement if argument n != x  
!(n=x) = skip over following statement if argument n == x  
?(n@y=x) = skip over following statement if argument n@y == x  
!(n@y=x) = skip over following statement if argument n@y != x  
 @y = represents a simple integer operation with another number y  
 Operations can be following: +, -, \*, /, %, <<, >>, &, |, ^

# Implementation

Sound isn’t generated directly from the soundstring, instead, the soundstring is converted to a simple sequence of statements, that say when a new tone should start playing and when it should stop. It defines all individual tones, gathered from all threads, defines its frequency (using a floating point number, so no shifts have to be decoded and applied), as well as soundfont and waveform. It also specifies the duration as floating point number.

This intermediary format is called ToneList and can be generated on its own, though usually it’s generated only internally and it’s converted to a sound file right away. ToneList can also optionally contain visualization data – that determines the position and length of character in the original soundstring that’s currently being played, however this part is purely optional and isn’t generated when it’s not needed, to improve performance and memory footprint.

ToneList consists of a string (which can be saved as a textual file), where each line represents one event, which can create several tones, but minimally one. Each line begins with a time in milliseconds (floating point numbers are allowed) of time when the event happens, therefore ToneList is always independent on the target samplerate – corresponding sample number is calculated when converting SoundList to a raw audio data.

ToneList syntax is made so it’s easily readable by human and can be edited or written manually, possibly also using alternate tools and languages and thus provide a way to extend the SoundString language without having to modify the sound generator.

A single ToneList entry has following syntax:

[event\_time]: [starttone0]; [starttone1]; … [starttoneN];

Each start tone event has following format:

[frequency\_HZ][waveform\_shape][volume\_float][volume\_shape][duration]”soundfont”[optinal\_visualization\_data]

Duration is in milliseconds and it’s useful only if given soundwave with corresponding soundfont needs it - if given soundfont supplies a constant waveform, which can fill the whole duration with a constant – repeated sounddata.

Optional visualization data have following format:

(startindex,endindex)

These specify indexes of the first and last characters in the original soundstring.

# Implementing the SoundString to SoundList

The implementation uses thread objects, which perform the parsing and executing of the commands in the string. The execution starts with a single thread at the beginning of the string, which will move trough the string, character by character and create more threads if needed, all of which will operate in parallel. Thread can wait for other threads to finish execution, usually when it plays a grouped statement, it waits for all of them to finish before moving to another one.

All

# Implementing the SoundList to sound data

Once SoundList is generated, it can be converted to a raw sound data: series of samples with configurable samplerate.

# Implemented Sounfonts

Soundsfonts can be reffered to by numbers, including when someone creates a soundfont and uploads it to the library of soundfonts – information system automatically generates a number that can be used to fetch the soundfont online, which can shorten the the actual soundstring, if a long name is supplied. The numbers used are in base ??? to make them as short as possible and still representable by printable ascii characters.

However, certain soundfonts are implemented directly in the engine and can thus be used right away. There are also reserved ranges of numbers that cannot be used for any user soundfonts. The scheme used is following and depends on the digits:

* One digit – reserved for internal soundfonts, never can be generated by user and never will be loaded from external file (it simply won’t even look if there’s such external file)
* Two digits – reserved for selected high quality soundfonts that are likely to be used very often
* Three digits and more – any user generated soundfont. It is unlikely for these to surpass more than three digits, because with given numeric base, a very large amount of combinations is possible