CSound Generator User Guide

Overview

The purpose of the CSound Generator (CSG) is to simplify generating large numbers of waveform definitions. The power of CSound is the level of control it provides over many aspects of a waveform. But that power involves complexity. If a composition contains a significant amount of repetition, then automating the generation of duplicate waveforms allows for large numbers of them, which leads to more interesting compositions.

It is strongly recommended that the CSound Book, edited by Richard Boulanger be used for reference. This comes with several examples for each of 30 chapters, but also many compositions by users of the CSound program which include the orchestra and score files.

Additionally, the CSound Canonical Reference should be included with the installation of the program itself. This provides the syntax information and examples of use for all CSound definitions.

The types of waveform repetition that can be implemented using the CSG include:

- Instrument harmonics
- Scale notes
- Staves
- Rhythm patterns

With these capabilities, the CSG is a tool for composition, but also analysis. CSound libraries provide standard synthesizer waveforms such as square, triangle, and sawtooth. But real world instruments have complex harmonics that are a mixture of the sound origin, the body, and human interaction (ie. plucking strings, striking drumheads with sticks, or blowing into a wind instrument). This means that many waveforms must be generated to sound realistic.

Scale notes and chords are the primary tools of music composition and are frequently repeated in most pieces. By defining the frequency and harmonics and automatically generating each instrument's notes, a great deal of effort is saved.

Rhythm patterns tend to be very repetitious, even when employing polyrhythms. The CSG has a REPEAT keyword to allow these to be repeated as often as needed.

One of the main reasons for developing this application was to study and compose microtonal music. The CSound Scale Notes Generator (CSSN) is included with the package to provide for generation of all possible scales in a given number of notes per octave, which means that it can be used to analyze scales within microtonal tunings.

Interface with CSound

CSound uses two configuration files, the Score file and the Orchestra file. The Orchestra file is used to define instruments and is relatively static. Once the instruments are defined, they can be left alone. The Score file feeds waveform information to the Orchestra file to produce the variation of a CSound composition. Therefore, the CSG creates only Score file definitions.

Once the Score file is created, CSound can be run, typically with the command:

```
csound --wave --output=<audio_filename> csg.orc csg.sco
```

The instrument definitions in the CSG template file must match one-for-one with those in the Orchestra file. However, many of these Orchestra file instrument definitions can be duplicates with unique IDs. Then the CSG template file is configured to create unique harmonic structures.

CSound Generator Definitions

There are three files used to configure CSG scores. In all files, comments are lines that have '#' in the first column. Blank lines are ignored.

- Notes file: This simply defines the frequencies of the notes to be used
- Template file: This is where instruments are defined
- Staves file: This is where the composition is created by defining notes to be played

Notes File

Note definitions have the following fields:

NOTE <name> <frequency>

- name is created with 3 characters as follows:
  - octave = 0 - 9
  - note: A - Z and a - z case sensitive
  - modifiers: Blank, 0 – 9, or A - Z and a - z case sensitive
- frequency is a floating point number

This allows for 52 notes and 63 modifiers which means up to 3,276 notes per octave can be defined. The frequencies of the octaves can range from about 20 Hz to nearly 30,000 Hz. But since this is controlled by the composer, it can be shifted up or down if desired.

Staves file

The staff definitions are also relatively simple. There are three required definitions:

STAFF <name>

- name is freeform and is entered as a comment in the CSound Score filename

INSTR <id>

- id is an integer that corresponds to an instrument in the Template file

One of FUND, TONE, or VFUND are required. Note that FUND and TONE are the same except for the volume field. Also, FUND and VFUND are the same except that VFUND includes a vowel identifier at the end.

FUND <note> <start> <duration> <volume>

- note matches an entry in the Notes file
- start is a floating point number that is the second relative to the beginning of the piece
- duration is a floating point number that defines how long the note is to be held
- volume is an integer between 1 and 32,000 which is the early method used by CSound to define volume

TONE <note> <start> <duration> <volume>

- note matches an entry in the Notes file
- start is a floating point number that is the second relative to the beginning of the piece
- duration is a floating point number that defines how long the note is to be held
- volume is a floating point number representing decibels

VFUND <note> <start> <duration> <volume> <vowel>

- note matches an entry in the Notes file
- start is a floating point number that is the second relative to the beginning of the piece
- duration is a floating point number that defines how long the note is to be held
- volume is an integer between 1 and 32,000 which is the early method used by CSound to define volume
- vowel is a string from the Template file VOWEL definition

There are 2 optional definitions:

TEMPO <multiplier>

- multiplier is a floating point number that is used to adjust the start and duration of notes either faster (< 1.0) or slower (> 1.0) than the specified times

  Note that in version 1.0 of the CSG, this affects the entire score, not just a single STAFF definition

REPEAT <count> <repeat time> <volume change> <period change>

- count is the number of time to repeat the following notes
- repeat time is a floating point number that is added to the start time of the pattern
- volume change is a positive or negative floating point number that is added to the volume of each note in the pattern
- period change is a positive or negative floating point number that is added to the repeat time for each repetition

Template File

The template definitions define instrument variable data. There are two sections the waveform functions and the instruments themselves. The definitions are explained in more detail in the next section. The syntax is as follows:

Function data defines waveforms in CSound. There are several GEN macros which can be used for this, but typically in CSG template files either GEN1 is used to create sine waves or GEN7 is used to create custom waves.

FUNC <function data>

- function data is freeform and is entered according to the CSound syntax

The INSTR definition is simply a number that corresponds to a definition in a CSound orchestra file. The orchestra file determines how the instrument waveforms are created by CSound. It is possible to create more instruments in the orchestra file than are in the template file, thus making it easier to experiment with instrument settings.

INSTR <id>

- id is an integer that corresponds to an instr definition in the orchestra file

The SIMP and TONE definitions are used one or more times to define the harmonics of an instrument. Notes from the Staves file are processed for each of these entries. This can result in a single line from a Staves file producing multiple lines in a Score file.

SIMP <start> <duration> <volume> <frequency multiplier> <remaining parameters>

- start is a floating point number that is added to the start value from the Staves file
- duration is a floating point number that is added to the duration value from the Staves file
- frequency multiplier is a floating point number that is multiplied with the frequency of the note being played (unless SCALEHARM is set, see below)
- volume is an integer that is divided by 1,000 and then added to the volume value from the Staves file
- remaining parameters depend on the instrument definition

TONE <start> <duration> <volume> <frequency multiplier> <remaining parameters>

- start is a floating point number that is added to the start value from the Staves file
- duration is a floating point number that is added to the duration value from the Staves file
- volume is a floating point number that is added to the volume value from the Staves file
- frequency multiplier is a floating point number that is multiplied with the frequency of the note being played (unless SCALEHARM is set, see below)
- remaining parameters depend on the instrument definition

The FIXED definition is used to set a note at a fixed frequency, regardless of the note definition from the Staves file. When combined with other notes for the instrument, this can produce modulation. It can also create the effect of human interaction with the instrument.

FIXED <start> <duration> <volume> <frequency> <remaining parameters>

- start is a floating point number that is added to the start value from the Staves file
- duration is a floating point number that is added to the duration value from the Staves file
- volume is an integer that is divided by 1,000 and then added to the volume value from the Staves file
- frequency is the fixed frequency for the note
- remaining parameters depend on the instrument definition

The SND definition is used to import audio files. These can be percussion, or Foley sounds. There should be a separate Notes file for these where the note names are different from the tuning notes and the frequency field is a division of low integers (ie. “NOTE 0X 0.25”, “NOTE 0Y 0.5”, “NOTE 0Z 0.75”, “NOTE 1W 1.0”, “NOTE 1X 1.25”, “NOTE 1Y 1.5”, “NOTE 1Z 1.75”NOTE 2X 2.25”, ...)

SND <start> <duration> <volume> <frequency multiplier>

- start is a floating point number that is added to the start value from the Staves file
- duration is a floating point number that is added to the duration value from the Staves file
- volume is an integer that is divided by 1,000 and then added to the volume value from the Staves file
- frequency multiplier is a floating point number that is multiplied times the audio wave being played to modify its frequency

The VOWEL definition allows strings to be mapped to the number used by the CSound fmvoice opcode. These are used to create a lookup table for the VOICE definition.

VOWEL <id> <name>

- id is the fmvoice opcode vowel parameter
- name is a descriptive name for the vowel sound of 1 - 15 characters

The VOICE definition is used in conjunction with the VOWEL definition to generate formant waves which make vowel sounds.

VOICE <start> <duration> <volume> <frequency multiplier> <id> <remaining parameters>

- start is a floating point number that is added to the start value from the Staves file
- duration is a floating point number that is added to the duration value from the Staves file
- frequency multiplier is a floating point number that is multiplied with the frequency of the note being played (unless SCALEHARM is set, see below)
- volume is an integer that is divided by 1,000 and then added to the volume value from the Staves file
- id is the number from the VOWEL table that matches the vowel name
- remaining parameters depend on the instrument definition

The SCALEHARM definition is a flag that indicates to use the frequency multiplier as an index into the list of notes relative to the note being played. Harmonics of 12 tone music are frequently whole number multiples of the fundamental. This allows harmonics to be notes from the microtonal tuning instead. Thus, in a 15 tone tuning, the octave would be 15, a note approximately and octave and a fifth would be 22, two octaves and a fourth would be 35, and so forth. If the fundamental plus the scale harmonic is beyond the range of the defined notes (ie. greater than 150 in a 15 tone tuning), it is ignored.

SCALEHARM

Command Line Parameters

The command line parameters for CSG are primarily used to set file names. The syntax is as follows:

```
csnd_gen [arguments]
csnd_genP2.py [arguments]   (Run under Python 2)
csnd_genP3.py [arguments]   (Run under Python 3)
-n: note_file: Name of note files, enter -n for each (Default: csg_notes)
-v: staves_file: Name of staves file (Default: csg_staves)
-t: template_file: Name of template file (Default: csg_tmplt)
-s: sco_file: Name of sco file (Default: csg.sco)
-d: debug code: Note = 1, Template = 2, Staff = 4 (Values can be ORed for multiple data types)
```

There may be multiple Notes files. This allows for both instrument notes and audio files for percussion to be used simultaneously. For example:

```
csnd_gen -n csg_notes.15tone -n csg_notes.sounds
```

However, there is also an interesting case where overlapping tunings can be used. In the following example, 16 tone, 20 tone, and 24 tone scales are blended. This requires that matching note in all files must have the same frequency and non-matching notes must have unique names. For example, in the command below, the files have the following definitions:

```
csg_notes.16tone:NOTE 0D 36.70
csg_notes.16tone:NOTE 0Dn 38.36
csg_notes.20tone:NOTE 0Df 35.46
csg_notes.20tone:NOTE 0D 36.70
csg_notes.20tone:NOTE 0Dm 38.02
csg_notes.24tone:NOTE 0Du 35.67
csg_notes.24tone:NOTE 0D 36.70
csg_notes.24tone:NOTE 0Do 37.80
csg_notes.24tone:NOTE 0Ds 38.90
```

The command to process these files is:

```
csnd_gen -n csg_notes.16tone -n csg_notes.20tone -n csg_24tone
```

This results in 47 unique notes per octave that are not all in an equidistant relationship (even logarithmic), but do have some notes in common with 12 tone scales.

Staves files can be named to identify what is in them, such as csg_staves.test, csg_staves.rhythm, or csg_staves.woodwinds. The command line argument for this is:

```
csnd_gen -v csg_staves.test -n csg_notes.15tone -n csg_sounds
```

A single template file, with its associated Orchestra file, can be created and copied from project to project. A few instruments might be changed, especially the audio files. But usually the changes will be additions as new instruments are desired for compositions.

CSound Scale Notes Generator Definitions

CSSN is a tool for analyzing microtonal scales. It finds all possible combinations of scales of a given length within a given tuning and creates CSG Staves files that can be converted to a CSound Score files to listen to the scales and compare them.

There are two files used to configure CSSN scores. In all files, comments are lines that have '#' in the first column. Blank lines are ignored.

- Notes file: This simply defines the list of the notes to be used
- Scale file: This provides a template for generating a Staves file with the scales selected by CSSN

Note that there must be csg_notes, csg_tmplt, and csg.orc files available for the CSG program to build the generated csg_staves files.

Notes File

Note definitions do not include the octave. The notes are put into a table and the size determines the tuning, or number of notes per octave. These definitions are as follows:

NOTE <name>

- name is created with characters as follows:
  - note: A - Z and a - z case sensitive
  - modifiers: Blank, 0 – 9, or A - Z and a - z case sensitive

Scale File

The Scale file is a template that looks like a Staves file with one exception. It is used to create multiple Staves files in order to hear and compare scales. There are three required definitions:

STAFF <name>

- name is freeform and this line is copied as is to a CSG Staves file

INSTR <id>

- id is an integer and this line is copied as is to a CSG Staves file

FUND <note> <start> <duration> <volume>

- note is a code that must be converted to an actual note name, the format is:
  - <octave>:<note index>
    - octave is 0 – 9
    - note index is the location in a list which starts at 0. Thus in the list, "A", "As", "B", "C', "Cs", the index of "A" is 0 and "Cs" is 4.
- start is a floating point number that is the second relative to the beginning of the piece
- duration is a floating point number that defines how long the note is to be held
- volume is an integer between 1 and 32,000

NEXT <interval>

- interval is a floating point number the defines the number of seconds between the start of one scale and the next if the '-g' option is set to group multiple scales in a Staves file

An example of a Scale file to play one 7 note scale is:

```
STAFF M1_1
INSTR 1
FUND 2:0  0.0   0.4   9000
FUND 2:1  0.12  0.4   9000
FUND 2:2  0.25  0.4   9000
FUND 2:3  0.37  0.4   9000
FUND 2:4  0.5   0.4   9000
FUND 2:5  0.62  0.4   9000
FUND 2:6  0.75  0.4   9000
```

Command Line Parameters

The command line parameters for CSSN are used to set file names and to filter the scales produced. Since there can be thousands of variations, the ability to limit the output to 10, 20, or even 50 simplifies the task of comparing them. The command syntax is as follows:

```
csnd_sn [arguments]
csnd_snP2.py [arguments]   (Run under Python 2)
csnd_snP3.py [arguments]   (Run under Python 3)
-n: note_file: Name of note file (Default: sn_notes)
-s: scale_file: Name of scale file (Default: sn_scale)
-v: staves_file: Name of staves file (Default: csg_staves)
-l: number: Length of scale excluding octave (Range: 1 - 99, Default: 7)
-i: number: Maximum interval (Default: 4=major 3rd, maximum: 25=fourteenth)
-f: number: First scale to generate (Default: 1, maximum: 8192)
-m: number: Maximum number of scales to generate (Default: all, maximum : 8192)
-x: string: Step pattern to exclude may be entered multiple times
   Examples: -x"hhh" -x"whw"
-g: number: Number of scales to group in a Staves file (Default: 1)
-k: Skip staves file generation
-d: debug code: Note = 1, Scales = 2, Staff = 4 (Values can be ORed for multiple data types)
```

The scale names used as extensions to the Staves files are the list of intervals in each selected scale where:

h = half tone, w = whole tone, m = minor 3rd, M = major 3rd, f = fourth,

d = diminished fifth, F = fifth, a = augmented fifth, s = sixth,

A = augmented sixth, S = seventh, j = major seventh, e = eighth,

n = diminished ninth, N = ninth, t = diminished tenth, T = tenth,

v = diminished eleventh, V = eleventh, l = diminished twelfth, L = twelfth,

r = diminished thirteenth, R = thirteenth, o = diminished fourteenth, O = fourteenth

Example: csg_staves.whMhmww

To differentiate these files from other Staves files, the ‘-v’ option can be used. The command to name the files sstaves.whMhmww is:

```
csnd_sn -v sstaves
```

The Note file and Scale file can be identified with an extension:

```
csnd_sn -n sn_notes.15tone -s sn_scale.15tone7note
```

As the number of notes in the tuning increase, the number of notes in scales can increase as well. And the intervals between notes can also become larger. The following command will generate 9 note scales (excluding the octave), with up to a five half tones between the notes in a 19 note tuning:

```
csnd_sn -n sn_notes.19tone -s sn_scale.19tone9note -l9 -i5
```

The number of unique scales becomes large fairly quickly. There are 222 unique scales of seven notes in a 15 tone tuning. There are 3,579 unique scales of 9 notes in a 19 note tuning with a maximum step size of 5. In order to limit these, or group them in blocks, the first and maximum arguments may be used:

```
csnd_sn -n sn_notes.19tone -s sn_scale.19tone9note -l9 -i5 -f200 -m25
```

However, it is even better to exclude some step patterns. The following are 9 note scales in a 19 tone tuning:

hhhhhhmff, hhhhhmhff, hhhhmhhff, hhhmhhhff, hmhhhhhff

Notice that essentially only 1 note changes between each scale. While one of these might be interesting, they are all going to sound virtually the same. This type of duplication can be excluded from the scales selected by using the ‘-x-’ option.

```
csnd_sn -n sn_notes.19tone -s sn_scale.19tone9note -l9 -i5 -x’hhh’ -x’www’
```

The group option allows multiple scales to be written to the same Staves file. The NEXT definition must be in the scale file to define the offset between the scales.

The combination of the skip option and the debug scales option allows the list of scales to be seen without generating any Staves files:

```
csnd_sn -n sn_notes.19tone -s sn_scale.19tone9note -l9 -i5 -x’hhh’ -x’www’ -d2 -k
```