sfz File Format

By: Cakewalk, *Development*
Filed Under: Specifications

The sfz Format: Basics

What is the sfz format?

The sfz format is a file format to define how a collection of samples are arragned for performance.

The goal behind the sfz format is to provide a free, simple, minimalistic and expandable format to arrange, distribute and use audio samples with the highest possible quality and the highest possible performance flexibility.

A sfz format file can be played in our freeware sfz player.

Soundware, software and hardware developers can create, use and distribute the sfz format files for free, for either free or commercial applications.

Some of the features of the sfz format are:

- Samples of any bit depth (8/16/24/32-bit) support, mono or stereo
- Samples taken at any samplerate (i.e. 44.1k, 48k, 88.2k, 96k, 176.4k, 192k, 384k)
- Compressed samples. Compressed and uncompressed can be combined
- Looped samples
- Unlimited keyboard splits and layers
- Unlimited velocity splits and layers
- Unlimited regions of sample playback based on MIDI controllers (continuous controllers, pitch bend, channel and polyphonic aftertouch, keyboard switches) and internal generators (random, sequence counters)
- Sample playback on MIDI control events
- Unlimited unidirectional and bidirectional exclusive regions (mute groups)
- Unlimited release trigger regions with release trigger attenuation control
- Unlimited crossfade controls
- Trigger on first-note and legato notes
- Sample playback synchronized to host tempo
- Dedicated Envelope Generators for pitch, filter and amplifier
- Dedicated LFO for pitch, filter and amplifier

How is the sfz format structured?

The sfz format is a collection of sample files plus one or multiple .sfz definition files. This structure, containing multiple files instead of a single file is defined as non-monolithic.

Two kinds of sample files were selected to be included in the sfz format: a basic PCM uncompressed format (standard Windows wave files) and a basic, adjustable-quality, royalty free compressed format (ogg-vorbis encoded files).

The inclusion of a compressed format allows sample developers and soundware creators to easily create preview or demonstration files in a small package so they can be transferred with minimum bandwidth, while retaining complete performance functionality.

Both formats are 100% royalty-free, so players can be created to reproduce them without fixed or per-copy fees. They can also be freely distributed on the web (provided that the contents of the files are copyright cleared).

Each .sfz definition file represents one or a collection of instruments. An instrument is defined as a collection of regions. Regions include the definition for the input controls, the samples (the wav/ogg files) and the performance parameters to play those samples.

How is the .sfz definition file created?

A .sfz definition file is just a text file. Consequently, it can be created by using any text editor (i.e. Notepad).

Why non-monolithic?

While both monolithic and non-monolithic formats have advantages and disadvantages, there are several reasons which moved us to adopt a non-monolithic sample format. Technological and conceptual reasons can hardly be separated, so here's a basic explanation.

The most important reason is the file size limitation of a non-monolitic file on FAT32 partitions. Samples are getting really big nowadays, with thousands of individual samples collected in single instruments, and triggered according to many input control combinations.

Samples with high bit resolution (i.e. 24-bit samples) and high samplerate settings (96kHz, 192kHz) make the collection size even bigger. In the case of a non-monolithic format, the limitation still applies, but it applies to each sample instead of to the sum of all samples, making the limit virtually unreachable.

While this limitation doesn't apply to NTFS, NTFS partitions are less efficient than FAT32 disks in terms of raw disk performance for streaming applications.

Additionally, editing a single sample in a monolithic file implies loading the whole file, and after edit, saving the whole file again to disk. When collection size is big, the loading and saving operation is very time-consuming.

However, we have not discharged the possibility of incorporating a monolithic format for the sfz format, as soon as the format structure is completely implemented. Small sound sets (or NTFS users) could chose between the two options appropriately.

Why not XML?

XML was actually the first choice for the .sfz definition file, mainly due the simplicity from the development point of view as the XML parser and transaction code is already available.

However, XML was designed to exchange data over the web. Musicians, players, composers, soundware developers and audio technicians generally do not know about XML at all.

In addition, as a universal information exchange format designed for general-purpose applications, XML is inefficient (in terms of information over total data terms), and editing a XML file requires the use of a XML editor instead of a text editor.

A .sfz file is extremely self-explanatory. Most of the functionality of an instrument can be easily discovered by reading the file.

Is there a .sfz dedicated editor?

From rgc:audio, not yet... and not anytime soon.
However, we're working with several developers in the industry, creators of sample-conversion software to implement the .sfz format in their converters and editors.

The nature of the format allows creating instruments using other general-purpose software, like spreadsheets, wordprocessors, simple-scripting languages and other custom tailored software applications.

Implementation

How is an instrument defined?

The basic component of an instrument is a region. An instrument then, is defined by one or more regions. Multiple regions can be arranged in a group. Groups allow entering common parameters for multiple regions.

A region can include three main components: the definition for a sample, a set of input controls and a set of performance parameters.

Sample

The sample opcode defines which sample file will be played when the region is defined to play.
If a sample opcode is not present in the region, the region will play the sample defined in the last <group>. If there's no previous group defined, or if the previous group doesn't specify a sample opcode, the region will be ignored.

Input Controls

Input controls define **when** the sample defined in a region will play, based in real-world controller values and/or internally calculated values.

Real-world controllers are the elements that players, musicians or composers actually use to play music. Internal values are calculated by the player, like sequence counters and random generators.

The sfz format relies in the standard Musical Instruments Digital Interface (MIDI) specification for all input controls. Most available performance controllers implement MIDI, and it's still the dominating specification for software audio sequencers in all platforms.

Keyboard controllers are the most significant example of an Input Controls generator. Other generators could be MIDI guitars and string instruments, wind controllers, drum and percussion controllers. With individual differences, they all generate a common set of messages defined in the MIDI specification.

A set of input controls then, are the combination of a played MIDI note with its velocity, continuous controllers, pitch bend, channel and polyphonic aftertouch, etc.

When a particular set of input controls matches the definition for a region, the sample specified in that region plays, using a particular set of performance parameters also specified in the region.

Inside the definition file, a region starts with the **<region>** header. A region is defined between two **<region>** headers, or between a **<region>** header and a **<group>** header, or between a **<region>** header and the end of the file.

Following the **<region>** header one or more opcodes can be defined. The opcodes are special keywords which instruct the player on what, when and how to play a sample.

Opcodes within a region can appear in any order, and they have to be separated by one or more spaces or tabulation controls. Opcodes can appear in separated lines within a region.

Opcodes and assigned opcode values are separated by the equal to sign (=), without spaces between the opcode and the sign. For instance:

sample=trombone_a4_ff.wav
sample=cello_a5_pp first take.wav

are valid examples, while:

sample = cello_a4_pp.wav

Is not (note the spaces at the sides of the = sign).
Input Controls and Performance Parameters opcodes are optional, so they might not be present in the definition file. An 'expectable' default value for each parameter is pre-defined, and will be used if there's no definition.

Example region definitions:

<region> sample=440.wav

This region definition instructs the player to play the sample file '440.wav' for the whole keyboard range.

<region> lokey=64 hikey=67 sample=440.wav

This region features a very basic set of input parameters (lokey and hikey, which represent the low and high MIDI notes in the keyboard), and the sample definition.
This instructs the player to play the sample '440.wav', if a key in the 64-67 range is played.

It is very important to note that all Input Controls defined in a region act using the AND boolean operator. Consequently, all conditions must be matched for the region to play. For instance:

<region> lokey=64 hikey=67 lovel=0 hivel=34 locc1=0 hicc1=40 sample=440.wav

This region definition instructs the player to play the sample '440.wav' if there is an incoming note event in the 64-67 range AND the note has a velocity in the 0~34 range AND last modulation wheel (cc1) message was in the 0~40 range.

Performance parameters

The Performance Parameters define **how** the sample specified will play, once the region is defined to play.
A simple example of a Performance Parameter is **volume**. It defines how loud the sample will be played when the region plays.

Groups

As previously stated, groups allow entering common parameters for multiple regions. A group is defined with the **<group>** opcode, and the parameters enumerated on it last till the next group opcode, or till the end of the file.

<group>
ampeg_attack=0.04 ampeg_release=0.45

<region> sample=trumpet_pp_c4.wav key=c4
<region> sample=trumpet_pp_c#4.wav key=c#4
<region> sample=trumpet_pp_d4.wav key=d4
<region> sample=trumpet_pp_d#4.wav key=d#4

<group>
<region> sample=trumpet_pp_e4.wav key=e4 // previous group parameters reset

Comments

Comment lines can be inserted anywhere inside the file. A comment line starts with the slash character ('/'), and it extends till the end of the line.

<region>
sample=trumpet_pp_c4.wav
// middle C in the keyboard
lokey=60
// pianissimo layer
lovel=0 hivel=20 // another comment

Where do the sample files have to be stored?

Sample files can be stored either in the same folder where the .sfz definition file resides, or in any alternative route, specified relatively to the location of the definition file. Consequently:

sample=trumpet_pp_c3.wav
sample=samples\trumpet_pp_c3.wav
sample=..\trumpet_pp_c3.wav

Are all valid sample names.

Alternatively, the player might specify one or several 'user folders', where it will search for samples if it doesn't find them in the same folder as the definition file.

What can the sfz format do?

The sfz format is aimed to allow the arrange of a sample collection in a flexible and expandable way. It's up to the player to decide which functionality it wants to implement.

Units

All units in the sfz format are in real-world values. Frequencies are expressed in Hertz, pitches in cents, amplitudes in percentage and volumes in decibels.
Notes are expressed in MIDI Note Numbers, or in note names according to the International Pitch Notation (IPN) convention. According to this rules, middle C in the keyboard is C4 and the MIDI note number 60.

Opcode List

The following is a description of all valid opcodes for the sfz format version 1.0:

Opcode	Description	Type	Default	Range
Sample Definition				
sample	This opcode defines which sample file the region will play. The value of this opcode is the filename of the sample file, including the extension. The filename must be stored in the same folder where the definition file is, or specified relatively to it. If the sample file is not found, the player will ignore the whole region contents. Long names and names with blank spaces and other special characters (excepting the = character) are allowed in the sample definition. The sample will play unchanged when a note equal to the **pitch_keycenter** opcode value is played. If **pitch_keycenter** is not defined for the region, sample will play unchanged on note 60 (middle C). Examples: sample=guitar_c4_ff.wav sample=dog kick.ogg sample=out of tune trombone (redundant).wav sample=staccatto_snare.ogg	string (filename)	n/a	n/a
Input Controls				
lochan **hichan**	If incoming notes have a MIDI channel between **lochan** and **hichan**, the region will play. Examples: lochan=1 hichan=5	integer	lochan=1 hichan=16	1 to 16

lokey
hikey
key
	If a note equal to or higher than **lokey** AND equal to or lower than **hikey** is played, the region will play.
	
	lokey and **hikey** can be entered in either MIDI note numbers (0 to 127) or in MIDI note names (C-1 to G9)
	
	The **key** opcode sets **lokey**, **hikey** and **pitch_keycenter** to the same note.
	
	Examples:
	lokey=60 // middle C
	hikey=63 // middle D#
	lokey=c4 // middle C
		integer	lokey=0, hikey=127	0 to 127
				C-1 to G9

	hikey=d#4 // middle D#
	hikey=eb4 // middle Eb (D#)			

lovel **hivel**	If a note with velocity value equal to or higher than **lovel** AND equal to or lower than **hivel** is played, the region will play.	integer	locc=0, hicc=127 for all controllers	0 to 127
lobend **hibend**	Defines the range of the last Pitch Bend message required for the region to play. Examples: lobend=0 hibend=4000 The region will play only if last Pitch Bend message received was in the 0~4000 range.	integer	lobend=-8192, hibend=8192	-8192 to 8192
lochanaft **hichanaft**	Defines the range of last Channel Aftertouch message required for the region to play. Examples: lochanaft=30 hichanaft=100 The region will play only if last Channel Aftertouch message received was in the 30~100 range.	integer	lochanaft=0, hichanaft=127	0 to 127
lopolyaft **hipolyaft**	Defines the range of last Polyphonic Aftertouch message required for the region to play. The incoming note information in the Polyphonic Aftertouch message is not relevant. Examples: lopolyaft=30 hipolyaft=100 The region will play only if last Polyphonic Aftertouch message received was in the 30~100 range.	integer	lopolyaft=0, hipolyaft=127	0 to 127
lorand **hirand**	Random values. The player will generate a new random number on every note-on event, in the range 0~1. The region will play if the random number is equal to or higher than lorand, and lower than hirand. Examples: lorand=0.2 hirand=0.4 lorand=0.4 hirand=1	floating point	lorand = 0 hirand = 1	0 to 1
lobpm **hibpm**	Host tempo value. The region will play if the host tempo is equal to or higher than lobpm, and lower than hibpm. Examples: lobpm=0 hibpm=100 lobpm=100 hibpm=200.5	floating point	lobpm = 0 hibpm = 500	0 to 500 bpm
seq_length	Sequence length. The player will keep an internal counter creating a consecutive note-on sequence for each region, starting at 1 and resetting at **seq_length**. Examples: seq_length=3	integer	1	1 to 100
seq_position	Sequence position. The region will play if the internal sequence counter is equal to **seq_position**. Examples: seq_length=4 seq_position=2 In above example, the region will play on the second note every four notes.	integer	1	1 to 100
sw_lokey **sw_hikey**	Defines the range of the keyboard to be used as trigger selectors for the **sw_last** opcode. **sw_lokey** and **sw_hikey** can be entered in either MIDI note numbers (0 to 127) or in MIDI note names (C-1 to G9) Examples: sw_lokey=48 sw_hikey=53	integer	sw_lokey=0, sw_hikey=127	0 to 127 C-1 to G9
sw_last	Enables the region to play if the last key pressed in the range specified by **sw_lokey** and **sw_hikey** is equal to the **sw_last** value. **sw_last** can be entered in either MIDI note numbers (0 to 127) or in MIDI note	integer	0	0 to 127

sw_last	names (C-1 to G9) Examples: sw_last=49	integer	0	C-1 to G9
sw_down	Enables the region to play if the key equal to **sw_down** value is depressed. Key has to be in the range specified by **sw_lokey** and **sw_hikey**. **sw_down** can be entered in either MIDI note numbers (0 to 127) or in MIDI note names (C-1 to G9) Examples: sw_down=Cb3	integer	0	0 to 127 C-1 to G9
sw_up	Enables the region to play if the key equal to **sw_up** value is not depressed. Key has to be in the range specified by **sw_lokey** and **sw_hikey**. sw_up can be entered in either MIDI note numbers (0 to 127) or in MIDI note names (C-1 to G9) Examples: sw_up=49	integer	0	0 to 127 C-1 to G9
sw_previous	Previous note value. The region will play if last note-on message was equal to **sw_previous** value. **sw_previous** can be entered in either MIDI note numbers (0 to 127) or in MIDI note names (C-1 to G9) Examples: sw_previous=60	integer	none	0 to 127 C-1 to G9
sw_vel	This opcode allows overriding the velocity for the region with the velocity of the previous note. Values can be: **current**: Region uses the velocity of current note. **previous**: Region uses the velocity of the previous note. Examples: sw_vel=previous	text	current	current, previous
trigger	Sets the trigger which will be used for the sample to play. Values can be: **attack** (default): Region will play on note-on. **release**: Region will play on note-off. The velocity used to play the note-off sample is the velocity value of the corresponding (previous) note-on message. **first**: Region will play on note-on, but if there's no other note going on (staccato, or first note in a legato phrase). **legato**: Region will play on note-on, but only if there's a note going on (notes after first note in a legato phrase). Examples: trigger=release	integer	attack	attack, release, first, legato
group	Exclusive group number for this region. Examples: group=3 group=334	integer	0	0 to 4Gb (4294967296)
off_by	Region off group. When a new region with a group number equal to **off_by** plays, this region will be turned off. Examples: off_by=3 off_by=334	integer	0	0 to 4Gb (4294967296)
off_mode	Region off mode. This opcode will determinate how a region is turned off by an **off_by** opcode. Values can be: **fast** (default): The voice will be turned off immediately. Release settings will not have any effect. **normal**: The region will be set into release stage. All envelope generators will enter in release stage, and region will expire when the amplifier envelope generator expired. Examples: off_mode=fast off_mode=normal	text	fast	fast, normal

	off_mode=normal			
on_loccN **on_hiccN**	Sample trigger on MIDI continuous control N. If a MIDI control message with a value between **on_loccN** and **on_hiccN** is received, the region will play. Examples: on_locc1=0 on_hicc1=0 Region will play when a MIDI CC1 (modulation wheel) message with zero value is received.	integer	-1 (unassigned)	0 to 127
Performance Parameters				
Sample Player				
delay	Region delay time, in seconds. If a **delay** value is specified, the region playback will be postponed for the specified time. If the region receives a note-off message before delay time, the region won't play. All envelope generators delay stage will start counting after region delay time. Examples: delay=1 delay=0.2	floating point	0	0 to 100 seconds
delay_random	Region random delay time, in seconds. If the region receives a note-off message before delay time, the region won't play. Examples: delay_random=1 delay_random=0.2	floating point	0	0 to 100 seconds
delay_ccN	Region delay time after MIDI continuous controller N messages are received, in seconds. If the region receives a note-off message before delay time, the region won't play. Examples: delay_cc1=1 delay_cc2=.5	floating point	0	0 to 100 seconds
offset	The offset used to play the sample, in sample units. The player will reproduce samples starting with the very first sample in the file, unless **offset** is specified. It will start playing the file at the **offset** sample in this case. Examples: offset=3000 offset=32425	integer	0	0 to 4 Gb (4294967296)
offset_random	Random offset added to the region offset, in sample units. Examples: offset_random=300 offset_random=100	integer	0	0 to 4 Gb (4294967296)
offset_ccN	The offset used to play the sample according to last position of MIDI continuous controller N, in sample units. This opcode is useful to specify an alternate sample start point based on MIDI controllers. Examples: offset_cc1=3000 offset_cc64=1388	integer	0	0 to 4 Gb (4294967296)
end	The endpoint of the sample, in sample units. The player will reproduce the whole sample if end is not specified. If end value is -1, the sample will not play. Marking a region end with -1 can be used to use a silent region to turn off other regions by using the group and off_by opcodes. Examples: end=133000 end=4432425	integer	0	-1 to 4 Gb (4294967296)
count	The number of times the sample will be played. If this opcode is specified, the sample will restart as many times as defined. Envelope generators will not be retriggered on sample restart. When this opcode is defined, loopmode is automatically set to one_shot. Examples:	integer	0	0 to 4 Gb (4294967296)

	count=3 count=2			
loop_mode	If **loop_mode** is not specified, each sample will play according to its predefined loop mode. That is, the player will play the sample looped using the first defined loop, if available. If no loops are defined, the wave will play unlooped. The **loop_mode** opcode allows playing samples with loops defined in the unlooped mode. The possible values are: ***no_loop*:** no looping will be performed. Sample will play straight from start to end, or until note off, whatever reaches first. ***one_shot*:** sample will play from start to end, ignoring note off. This mode is engaged automatically if the **count** opcode is defined. ***loop_continuous*:** once the player reaches sample loop point, the loop will play until note expiration. ***loop_sustain*:** the player will play the loop while the note is held, by keeping it depressed or by using the sustain pedal (CC64). The rest of the sample will play after note release. Examples: loop_mode=no_loop loop_mode=loop_continuous	text	**no_loop** for samples without a loop defined, **loop_continuous** for samples with defined loop(s).	n/a
loop_start	The loop start point, in samples. If **loop_start** is not specified and the sample has a loop defined, the sample start point will be used. If **loop_start** is specified, it will overwrite the loop start point defined in the sample. This opcode will not have any effect if loopmode is set to **no_loop**. Examples: loop_start=4503 loop_start=12445	integer	0	0 to 4 Gb (4294967296)
loop_end	The loop end point, in samples. This opcode will not have any effect if loopmode is set to **no_loop**. If **loop_end** is not specified and the sample have a loop defined, the sample loop end point will be used. If **loop_end** is specified, it will overwrite the loop end point defined in the sample. Examples: loop_end=34503 loop_end=212445	integer	0	0 to 4 Gb (4294967296)
sync_beats	Region playing synchronization to host position. When **sync_beats** is specified and after input controls instruct the region to play, the playback will be postponed until the next multiple of the specified value is crossed. Examples: sync_beats=4 In this example, if note is pressed in beat 2 of current track, note won't be played until beat 4 reaches. This opcode will only work in hosts featuring song position information (vstTimeInfo ppqPos).	floating point	0	0 to 32 beats
sync_offset	Region playing synchronization to host position offset. When **sync_beats** is specified and after input controls instruct the region to play, the playback will be postponed until the next multiple of the specified value plus the **sync_offset** value is crossed. Examples: sync_beats=4 sync_offset=1 In this example, if note is pressed in beat 2 of current track, note won't be played until beat 5 reaches. This opcode will only work in hosts featuring song position information (vstTimeInfo ppqPos).	floating point	0	0 to 32 beats
Pitch				
transpose	The transposition value for this region which will be applied to the sample. Examples: transpose=3 transpose=-4	integer	0	-127 to 127
tune	The fine tuning for the sample, in cents. Range is Â±1 semitone, from -100 to 100. Only negative values must be prefixed with sign. Examples: tune=33 tune=-30 tune=94	integer	0	-100 to 100

	tune=94			
pitch_keycenter	Root key for the sample. Examples: pitch_keycenter=56 pitch_keycenter=c#2	integer	60 (C4)	-127 to 127 C-1 to G9
pitch_keytrack	Within the region, this value defines how much the pitch changes with every note. Default value is 100, which means pitch will change one hundred cents (one semitone) per played note. Setting this value to zero means that all notes in the region will play the same pitch, particularly useful when mapping drum sounds. Examples: pitch_keytrack=20 pitch_keytrack=0	integer	100	-1200 to 1200
pitch_veltrack	Pitch velocity tracking, represents how much the pitch changes with incoming note velocity, in cents. Examples: pitch_veltrack=0 pitch_veltrack=1200	integer	0	-9600 to 9600 cents
pitch_random	Random tuning for the region, in cents. Random pitch will be centered, with positive and negative values. Examples: pitch_random=100 pitch_random=400	integer	0	0 to 9600 cents
bend_up	Pitch bend range when Bend Wheel or Joystick is moved up, in cents. Examples: bend_up=1200 bend_up=100	integer	200	-9600 to 9600
bend_down	Pitch bend range when Bend Wheel or Joystick is moved down, in cents. Examples: bend_down=1200 bend_down=100	integer	-200	

bend_step	Pitch bend step, in cents.
	
	Examples:
	bend_step=100 // glissando in semitones
	bend_step=200 // glissando in whole tones
		integer	1	1 to 1200

Pitch EG				
pitcheg_delay	Pitch EG delay time, in seconds. This is the time elapsed from note on to the start of the Attack stage. Examples: pitcheg_delay=1.5 pitcheg_delay=0	floating point	0 seconds	0 to 100 seconds
pitcheg_start	Pitch EG start level, in percentage. Examples: pitcheg_start=20 pitcheg_start=100	floating point	0 %	0 to 100 %
pitcheg_attack	Pitch EG attack time, in seconds. Examples: pitcheg_attack=1.2 pitcheg_attack=0.1	floating point	0 seconds	0 to 100 seconds
pitcheg_hold	Pitch EG hold time, in seconds. During the hold stage, EG output will remain at its maximum value. Examples: pitcheg_hold=1.5 pitcheg_hold=0.1	floating point	0 seconds	0 to 100 seconds
pitcheg_decay	Pitch EG decay time, in seconds. Examples: pitcheg_decay=1.5 pitcheg_decay=3	floating point	0 seconds	0 to 100 seconds
pitcheg_sustain	Pitch EG release time (after note release), in seconds. Examples: pitcheg_release=1.34 pitcheg_release=2	floating point	100 %	0 to 100 %
pitcheg_release	Pitch EG release time (after note release), in seconds. Examples: pitcheg_release=1.34 pitcheg_release=2	floating point	0 seconds	0 to 100 seconds
	Depth for the pitch EG, in cents.			

pitcheg_depth	Examples: pitcheg_depth=1200 pitcheg_depth=-100	integer	0	-12000 to 12000
pitcheg_vel2delay	Velocity effect on pitch EG delay time, in seconds. Examples: pitcheg_vel2delay=1.2 pitcheg_vel2delay=0.1 Delay time will be calculated as **delay time = pitcheg_delay + pitcheg_vel2delay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
pitcheg_vel2attack	Velocity effect on pitch EG attack time, in seconds. Examples: pitcheg_vel2attack=1.2 pitcheg_vel2attack=0.1 Attack time will be calculated as **attack time = pitcheg_attack + pitcheg_vel2attack * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
pitcheg_vel2hold	Velocity effect on pitch EG hold time, in seconds. Examples: pitcheg_vel2hold=1.2 pitcheg_vel2hold=0.1 Hold time will be calculated as **hold time = pitcheg_hold + pitcheg_vel2hold * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
pitcheg_vel2decay	Velocity effect on pitch EG decay time, in seconds. Examples: pitcheg_vel2decay=1.2 pitcheg_vel2decay=0.1 Decay time will be calculated as **decay time = pitcheg_decay + pitcheg_vel2decay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
pitcheg_vel2sustain	Velocity effect on pitch EG sustain level, in percentage. Examples: pitcheg_vel2sustain=30 pitcheg_vel2sustain=20 Sustain level will be calculated as **sustain level = pitcheg_sustain + pitcheg_vel2sustain**	floating point	0 %	-100 % to 100 %
pitcheg_vel2release	Velocity effect on pitch EG release time, in seconds. Examples: pitcheg_vel2release=1.2 pitcheg_vel2release=0.1 Release time will be calculated as **release time = pitcheg_release + pitcheg_vel2release * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
pitcheg_vel2depth	Velocity effect on pitch EG depth, in cents. Examples: pitcheg_vel2depth=100 pitcheg_vel2depth=-1200	integer	0 cents	-12000 to 12000 cents
Pitch LFO				
pitchlfo_delay	The time before the Pitch LFO starts oscillating, in seconds. Examples: pitchlfo_delay=1 pitchlfo_delay=0.4	floating point	0 seconds	0 to 100 seconds
pitchlfo_fade	Pitch LFO fade-in effect time. Examples: pitchlfo_fade=1 pitchlfo_fade=0.4	floating point	0 seconds	0 to 100 seconds
pitchlfo_freq	Pitch LFO frequency, in hertz. Examples: pitchlfo_freq=0.4 pitchlfo_freq=1.3	floating point	0 Hertz	0 to 20 hertz
pitchlfo_depth	Pitch LFO depth, in cents. Examples: pitchlfo_depth=1 pitchlfo_depth=4	integer	0 cent	-1200 to 1200 cents
pitchlfo_depthccN	Pitch LFO depth when MIDI continuous controller N is received, in cents. Examples:	integer	0 cent	-1200 to 1200

pitchlfo_depthccN	Examples: pitchlfo_depthcc1=100 pitchlfo_depthcc32=400	integer	0 cent	cents
pitchlfo_depthchanaft	Pitch LFO depth when channel aftertouch MIDI messages are received, in cents. Examples: pitchlfo_depthchanaft=100 pitchlfo_depthchanaft=400	integer	0 cent	-1200 to 1200 cents
pitchlfo_depthpolyaft	Pitch LFO depth when polyphonic aftertouch MIDI messages are received, in cents. Examples: pitchlfo_depthpolyaft=100 pitchlfo_depthpolyaft=400	integer	0 cent	-1200 to 1200 cents
pitchlfo_freqccN	Pitch LFO frequency change when MIDI continuous controller N is received, in hertz. Examples: pitchlfo_freqcc1=5 pitchlfo_freqcc1=-12	floating point	0 hertz	-200 to 200 hertz
pitchlfo_freqchanaft	Pitch LFO frequency change when channel aftertouch MIDI messages are received, in hertz. Examples: pitchlfo_freqchanaft=10 pitchlfo_freqchanaft=-40	floating point	0 hertz	-200 to 200 hertz
pitchlfo_freqpolyaft	Pitch LFO frequency change when polyphonic aftertouch MIDI messages are received, in hertz. Examples: pitchlfo_freqpolyaft=10 pitchlfo_freqpolyaft=-4	floating point	0 hertz	-200 to 200 hertz
Filter				
fil_type	Filter type. Avaliable types are: **lpf_1p**: one-pole low pass filter (6dB/octave). **hpf_1p**: one-pole high pass filter (6dB/octave). **lpf_2p**: two-pole low pass filter (12dB/octave). **hpf_2p**: two-pole high pass filter (12dB/octave). **bpf_2p**: two-pole band pass filter (12dB/octave). **brf_2p**: two-pole band rejection filter (12dB/octave). Examples: fil_type=lpf_2p fil_type=hpf_1p	text	lpf_2p	lpf_1p, hpf_1p, lpf_2p, hpf_2p, bpf_2p, brf_2p
cutoff	The filter cutoff frequency, in Hertz. If the cutoff is not specified, the filter will be disabled, with the consequent CPU drop in the player. Examples: cutoff=343 cutoff=4333	floating point	filter disabled	0 to SampleRate / 2
cutoff_ccN	The variation in the cutoff frequency when MIDI continuous controller N is received, in cents. Examples: cutoff_cc1=1200 cutoff_cc2=-100	integer	0	-9600 to 9600 cents
cutoff_chanaft	The variation in the cutoff frequency when MIDI channel aftertouch messages are received, in cents. Examples: cutoff_chanaft=1200 cutoff_chanaft=-100	integer	0	-9600 to 9600 cents
cutoff_polyaft	The variation in the cutoff frequency when MIDI polyphonic aftertouch messages are received, in cents. Examples: cutoff_polyaft=1200 cutoff_polyaft=-100	integer	0	-9600 to 9600 cents
resonance	The filter cutoff resonance value, in decibels. Examples: resonance=30	floating point	0 dB	0 to 40 dB
fil_keytrack	Filter keyboard tracking (change on cutoff for each key) in cents. Examples: fil_keytrack=100 fil_keytrack=0	integer	0 cents	0 to 1200 cents
fil_keycenter	Center key for filter keyboard tracking. In this key, the filter keyboard tracking will have no effect. Examples: fil_keycenter=60	integer	60	0 to 127

	fil_keycenter=60 fil_keycenter=48			
fil_veltrack	Filter velocity tracking, represents how much the cutoff changes with incoming note velocity. Examples: fil_veltrack=0 fil_veltrack=1200	integer	0	-9600 to 9600 cents
fil_random	Random cutoff added to the region, in cents. Examples: fil_random=100 fil_random=400	integer	0	0 to 9600 cents
Filter EG				
fileg_delay	Filter EG delay time, in seconds. This is the time elapsed from note on to the start of the Attack stage. Examples: fileg_delay=1.5 fileg_delay=0	floating point	0 seconds	0 to 100 seconds
fileg_start	Filter EG start level, in percentage. Examples: fileg_start=20 fileg_start=100	floating point	0 %	0 to 100 %
fileg_attack	Filter EG attack time, in seconds. Examples: fileg_attack=1.2 fileg_attack=0.1	floating point	0 seconds	0 to 100 seconds
fileg_hold	Filter EG hold time, in seconds. During the hold stage, EG output will remain at its maximum value. Examples: fileg_hold=1.5 fileg_hold=0.1	floating point	0 seconds	0 to 100 seconds
fileg_decay	Filter EG decay time, in seconds. Examples: fileg_decay=1.5 fileg_decay=3	floating point	0 seconds	0 to 100 seconds
fileg_sustain	Filter EG sustain level, in percentage. Examples: fileg_sustain=40.34 fileg_sustain=10	floating point	100 %	0 to 100 %
fileg_release	Filter EG release time (after note release), in seconds. Examples: fileg_release=1.34 fileg_release=2	floating point	0 seconds	0 to 100 seconds
fileg_depth	Depth for the filter EG, in cents. Examples: fileg_depth=1200 fileg_depth=-100	integer	0	-12000 to 12000
fileg_vel2delay	Velocity effect on filter EG delay time, in seconds. Examples: fileg_vel2delay=1.2 fileg_vel2delay=0.1 Delay time will be calculated as **delay time = fileg_delay + fileg_vel2delay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
fileg_vel2attack	Velocity effect on filter EG attack time, in seconds. Examples: fil_vel2attack=1.2 fil_vel2attack=0.1 Attack time will be calculated as **attack time = fileg_attack + fileg_vel2attack * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
fileg_vel2hold	Velocity effect on filter EG hold time, in seconds. Examples: fileg_vel2hold=1.2 fileg_vel2hold=0.1 Hold time will be calculated as **hold time = fileg_hold + fileg_vel2hold * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
	Velocity effect on filter EG decay time, in seconds. Examples:			

fileg_vel2decay	fileg_vel2decay=1.2 fileg_vel2decay=0.1 Decay time will be calculated as **decay time = fileg_decay + fileg_vel2decay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
fileg_vel2sustain	Velocity effect on filter EG sustain level, in percentage. Examples: fileg_vel2sustain=30 fileg_vel2sustain=-30 Sustain level will be calculated as **sustain level = fileg_sustain + fileg_vel2sustain** Result will be clipped to 0~100%.	floating point	0 %	-100 % to 100 %
fileg_vel2release	Velocity effect on filter EG release time, in seconds. Examples: fileg_vel2release=1.2 fileg_vel2release=0.1 Release time will be calculated as **release time = fileg_release + fileg_vel2release * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
fileg_vel2depth	-12000 to 12000 cents	integer	0 cents	-12000 to 12000 cents
Filter LFO				
fillfo_delay	The time before the filter LFO starts oscillating, in seconds. Examples: fillfo_delay=1 fillfo_delay=0.4	floating point	0 seconds	0 to 100 seconds
fillfo_fade	Filter LFO fade-in effect time. Examples: fillfo_fade=1 fillfo_fade=0.4	floating point	0 seconds	0 to 100 seconds
fillfo_freq	Filter LFO frequency, in hertz. Examples: fillfo_freq=0.4 fillfo_freq=1.3	floating point	0 Hertz	0 to 20 hertz
fillfo_depth	Filter LFO depth, in cents. Examples: fillfo_depth=1 fillfo_depth=4	floating point	0 dB	-1200 to 1200 cents
fillfo_depthccN	Filter LFO depth when MIDI continuous controller N is received, in cents. Examples: fillfo_depthcc1=100 fillfo_depthcc32=400	integer	0 cent	-1200 to 1200 cents
fillfo_depthchanaft	Filter LFO depth when channel aftertouch MIDI messages are received, in cents. Examples: fillfo_depthchanaft=100 fillfo_depthchanaft=400	integer	0 cent	-1200 to 1200 cents
fillfo_depthpolyaft	Filter LFO depth when polyphonic aftertouch MIDI messages are received, in cents. Examples: fillfo_depthpolyaft=100 fillfo_depthpolyaft=400	integer	0 cent	-1200 to 1200 cents
fillfo_freqccN	Filter LFO frequency change when MIDI continuous controller N is received, in hertz. Examples: fillfo_freqcc1=5 fillfo_freqcc1=-12	floating point	0 hertz	-200 to 200 hertz
fillfo_freqchanaft	Filter LFO frequency change when channel aftertouch MIDI messages are received, in hertz. Examples: fillfo_freqchanaft=10 fillfo_freqchanaft=-40	floating point	0 hertz	-200 to 200 hertz
fillfo_freqpolyaft	Filter LFO frequency change when polyphonic aftertouch MIDI messages are received, in hertz. Examples: fillfo_freqpolyaft=10 fillfo_freqpolyaft=-4	floating point	0 hertz	-200 to 200 hertz
Amplifier				
volume	The volume for the region, in decibels. Examples: volume=-24	floating point	0.0	-144 to 6 dB

	volume=0 volume=3.5			
pan	The panoramic position for the region. If a mono sample is used, **pan** value defines the position in the stereo image where the sample will be placed. When a stereo sample is used, the pan value the relative amplitude of one channel respect the other. A value of zero means centered, negative values move the panoramic to the left, positive to the right. Examples: pan=-30.5 pan=0 pan=43	floating point	0.0	-100 to 100 %

width	Only operational for stereo samples, **width** defines the amount of channel mixing applied to play the sample.
	
	A **width** value of 0 makes a stereo sample play as if it were mono (adding both channels and compensating for the resulting volume change). A value of 100 will make the stereo sample play as original.
	
	Any value in between will mix left and right channels with a part of the other, resulting in a narrower stereo field image.
	
	Negative **width** values will reverse left and right channels.
	
	Examples:
	width=100 // stereo
	width=0 // play this stereo sample as mono
	width=50 // mix 50% of one channel with the other
		floating point	0.0	-100 to 100 %

position	Only operational for stereo samples, **position** defines the position in the stereo field of a stereo signal, after channel mixing as defined in the **width** opcode.
	
	A value of zero means centered, negative values move the panoramic to the left, positive to the right.
	
	Examples:
	// mix both channels and play the result at left
	
	width=0 position=-100
	
	// make the stereo image narrower and play it
	// slightly right
	width=50 position=30
		floating point	0.0	-100 to 100 %

amp_keytrack	Amplifier keyboard tracking (change in amplitude per key) in dB. Examples: amp_keytrack=-1.4 amp_keytrack=3	floating point	0 dB	-96 to 12 dB
amp_keycenter	Center key for amplifier keyboard tracking. In this key, the amplifier keyboard tracking will have no effect. Examples: amp_keycenter=60 amp_keycenter=48	integer	60	0 to 127
amp_veltrack	Amplifier velocity tracking, represents how much the amplitude changes with incoming note velocity. Volume changes with incoming velocity in a concave shape according to the following expression: Amplitude(dB) = 20 log (127^2 / Velocity^2) The **amp_velcurve_N** opcodes allow overriding the default velocity curve. Examples: amp_veltrack=0 amp_veltrack=100	floating point	100 %	-100 to 100 %

amp_velcurve_1
amp_velcurve_127
	User-defined amplifier velocity curve. This opcode range allows defining a specific curve for the amplifier velocity. The value of the opcode indicates the normalized amplitude (0 to 1) for the specified velocity.
	
	The player will interpolate lineraly between specified opcodes for unspecified ones:
	
	amp_velcurve_1=0.2 amp_velcurve_3=0.3
	// amp_velcurve_2 is calculated to 0.25
	
	If **amp_velcurve_127** is not specified, the player will assign it the value of 1.
	
	Examples:
	// linear, compressed dynamic range
	// amplitude changes from 0.5 to 1
	amp_velcurve_1=0.5
		floating point	standard curve (see **amp_veltrack**)	0 to 1

amp_random	Random volume for the region, in decibels. Examples: amp_random=10 amp_random=3	floating point	0	0 to 24 dB
rt_decay	The volume decay amount when the region is set to play in **release** trigger mode, in decibels per second since note-on message. Examples: rt_decay=6.5	floating point	0 dB	0 to 200 dB
output	The stereo output number for this region. If the player doesn't feature multiple outputs, this opcode is ignored. Examples: output=0 output=4	integer	0	0 to 1024
gain_ccN	Gain applied on MIDI control N, in decibels. Examples: gain_cc1=12	floating point	0	-144 to 48 dB
xfin_lokey **xfin_hikey**	Fade in control. **xfin_lokey** and **xfin_hikey** define the fade-in keyboard zone for the region. The volume of the region will be zero for keys lower than or equal to **xfin_lokey**, and maximum (as defined by the **volume** opcode) for keys greater than or equal to **xfin_hikey.** Examples: xfin_lokey=c3 xfin_hikey=c4	integer	xfin_lokey=0 xfin_hikey=0	0 to 127 C-1 to G9
xfout_lokey **xfout_hikey**	Fade out control. **xfout_lokey** and **xfout_hikey** define the fade-out keyboard zone for the region. The volume of the region will be maximum (as defined by the **volume** opcode) for keys lower than or equal to **xfout_lokey**, and zero for keys greater than or equal to **xfout_hikey.** Examples: xfout_lokey=c5 xfout_hikey=c6	integer	xfout_lokey=127 xfout_hikey=127	0 to 127 C-1 to G9
xf_keycurve	Keyboard crossfade curve for the region. Values can be: **gain:** Linear gain crossfade. This setting is best when crossfading phase-aligned material. Linear gain crossfades keep constant amplitude during the crossfade, preventing clipping. **power:** Equal-power RMS crossfade. This setting works better to mix very different material, as a constant power level is kept during the crossfade.	text	power	gain, power
xfin_lovel **xfin_hivel**	Fade in control. **xfin_lovel** and **xfin_hivel** define the fade-in velocity range for the region. The volume of the region will be zero for velocities lower than or equal to **xfin_lovel**, and maximum (as defined by the **volume** opcode) for velocities greater than or equal to **xfin_hivel.** Examples: xfin_lovel=0 xfin_hivel=127	integer	xfin_lovel=0 xfin_hivel=0	0 to 127
xfout_lovel **xfout_hivel**	Fade out control. **xfout_lokey** and **xfout_hikey** define the fade-out velocity range for the region. The volume of the region will be maximum (as defined by the **volume** opcode) for velocities lower than or equal to **xfout_lovel**, and zero for velocities greater than or equal to **xfout_hivel.** Examples: xfout_lovel=0 xfout_hivel=127	integer	xfout_lokey=127 xfout_hikey=127	0 to 127
xf_velcurve	Velocity crossfade curve for the region. Values can be: **gain:** Linear gain crossfade. This setting is best when crossfading phase-aligned material. Linear gain crossfades keep constant amplitude during the crossfade, preventing clipping. **power:** Equal-power RMS crossfade. This setting works better to mix very different material, as a constant power level is kept during the crossfade.	text	power	gain, power

xfin_loccN **xfin_hiccN**	Fade in control. **xfin_loccN** and **xfin_hiccN** set the range of values in the MIDI continuous controller N which will perform a fade-in in the region. The volume of the region will be zero for values of the MIDI continuous controller N lower than or equal to **xfin_loccN**, and maximum (as defined by the **volume** opcode) for values greater than or equal to **xfin_hiccN.** Examples: xfin_locc1=64 xfin_hicc1=127	integer	0	0 to 127
xfout_loccN **xfout_hiccN**	Fade out control. **xfout_loccN** and **xfout_hiccN** set the range of values in the MIDI continuous controller N which will perform a fade-out in the region. The volume of the region will be maximum (as defined by the **volume** opcode) for values of the MIDI continuous controller N lower than or equal to **xfout_loccN**, and zero for values greater than or equal to **xfout_hiccN.** Examples: xfout_locc1=64 xfout_hicc1=127	integer	0	0 to 127
xf_cccurve	MIDI controllers crossfade curve for the region. Values can be: **gain:** Linear gain crossfade. This setting is best when crossfading phase-aligned material. Linear gain crossfades keep constant amplitude during the crossfade, preventing clipping. **power:** Equal-power RMS crossfade. This setting works better to mix very different material, as a constant power level is kept during the crossfade.	text	power	gain, power
Amplifier EG				
ampeg_delay	Amplifier EG delay time, in seconds. This is the time elapsed from note on to the start of the Attack stage. Examples: ampeg_delay=1.5 ampeg_delay=0	floating point	0 seconds	0 to 100 seconds
ampeg_start	Amplifier EG start level, in percentage. Examples: ampeg_start=20 ampeg_start=100	floating point	0 %	0 to 100 %
ampeg_attack	Amplifier EG attack time, in seconds. Examples: ampeg_attack=1.2 ampeg_attack=0.1	floating point	0 seconds	0 to 100 seconds
ampeg_hold	Amplifier EG hold time, in seconds. During the hold stage, EG output will remain at its maximum value. Examples: ampeg_hold=1.5 ampeg_hold=0.1	floating point	0 seconds	0 to 100 seconds
ampeg_decay	Amplifier EG decay time, in seconds. Examples: ampeg_decay=1.5 ampeg_decay=3	floating point	0 seconds	0 to 100 seconds
ampeg_sustain	Amplifier EG sustain level, in percentage. Examples: ampeg_sustain=40.34 ampeg_sustain=10	floating point	100 %	0 to 100 %
ampeg_release	Amplifier EG release time (after note release), in seconds. Examples: ampeg_release=1.34 ampeg_release=2	floating point	0 seconds	0 to 100 seconds
ampeg_vel2delay	Velocity effect on amplifier EG delay time, in seconds. Examples: ampeg_vel2delay=1.2 ampeg_vel2delay=0.1 Delay time will be calculated as **delay time = ampeg_delay + ampeg_vel2delay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
	Velocity effect on amplifier EG attack time, in seconds. Examples:			

ampeg_vel2attack	Examples: ampeg_vel2attack=1.2 ampeg_vel2attack=0.1 Attack time will be calculated as **attack time = ampeg_attack + ampeg_vel2attack * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
ampeg_vel2hold	Velocity effect on amplifier EG hold time, in seconds. Examples: ampeg_vel2hold=1.2 ampeg_vel2hold=0.1 Hold time will be calculated as **hold time = ampeg_hold + ampeg_vel2hold * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
ampeg_vel2decay	Velocity effect on amplifier EG decay time, in seconds. Examples: ampeg_vel2decay=1.2 ampeg_vel2decay=0.1 Decay time will be calculated as **decay time = ampeg_decay + ampeg_vel2decay * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
ampeg_vel2sustain	Velocity effect on amplifier EG sustain level, in percentage. Examples: ampeg_vel2sustain=30 ampeg_vel2sustain=-30 Sustain level will be calculated as **sustain level= ampeg_sustain + ampeg_vel2sustain** The result will be clipped to 0~100%.	floating point	0%	-100 % to 100 %
ampeg_vel2release	Velocity effect on amplifier EG release time, in seconds. Examples: ampeg_vel2release=1.2 ampeg_vel2release=0.1 Release time will be calculated as **release time = ampeg_release + ampeg_vel2release * velocity / 127**	floating point	0 seconds	-100 to 100 seconds
ampeg_delayccN	Amplifier EG delay time added on MIDI control N, in seconds. Examples: ampeg_delaycc20=1.5 ampeg_delaycc1=0	floating point	0 seconds	-100 to 100 seconds
ampeg_startccN	Amplifier EG start level added on MIDI control N, in percentage. Examples: ampeg_startcc20=20 ampeg_startcc1=100	floating point	0 %	-100 to 100 %
ampeg_attackccN	Amplifier EG attack time added on MIDI control N, in seconds. Examples: ampeg_attackcc20=1.2 ampeg_attackcc1=0.1	floating point	0 seconds	-100 to 100 seconds
ampeg_holdccN	Amplifier EG hold time added on MIDI control N, in seconds. Examples: ampeg_holdcc20=1.5 ampeg_holdcc1=0.1	floating point	0 seconds	-100 to 100 seconds
ampeg_decayccN	Amplifier EG decay time added on MIDI control N, in seconds. Examples: ampeg_decaycc20=1.5 ampeg_decaycc1=3	floating point	0 seconds	-100 to 100 seconds
ampeg_sustainccN	Amplifier EG sustain level added on MIDI control N, in percentage. Examples: ampeg_sustaincc20=40.34 ampeg_sustaincc1=10	floating point	100 %	-100 to 100 %
ampeg_releaseccN	Amplifier EG release time added on MIDI control N, in seconds. Examples: ampeg_releasecc20=1.34 ampeg_releasecc1=2	floating point	0 seconds	-100 to 100 seconds
Amplifier LFO				
amplfo_delay	The time before the Amplifier LFO starts oscillating, in seconds. Examples:	floating point	0 seconds	0 to 100 seconds

	amplfo_delay=1 amplfo_delay=0.4			
amplfo_fade	Amplifier LFO fade-in effect time. Examples: amplfo_fade=1 amplfo_fade=0.4	floating point	0 seconds	0 to 100 seconds
amplfo_freq	Amplifier LFO frequency, in hertz. Examples: amplfo_freq=0.4 amplfo_freq=1.3	floating point	0 Hertz	0 to 20 hertz
amplfo_depth	Amplifier LFO depth, in decibels. Examples: amplfo_depth=1 amplfo_depth=4	floating point	0 dB	-10 to 10 dB
amplfo_depthccN	Amplifier LFO depth when MIDI continuous controller N is received, in decibels. Examples: amplfo_depthcc1=100 amplfo_depthcc32=400	floating point	0 dB	-10 to 10 dB
amplfo_depthchanaft	Amplifier LFO depth when channel aftertouch MIDI messages are received, in cents. Examples: amplfo_depthchanaft=100 amplfo_depthchanaft=400	floating point	0 dB	-10 to 10 dB
amplfo_depthpolyaft	Amplifier LFO depth when polyphonic aftertouch MIDI messages are received, in cents. Examples: amplfo_depthpolyaft=100 amplfo_depthpolyaft=400	floating point	0 dB	-10 to 10 dB
amplfo_freqccN	Amplifier LFO frequency change when MIDI continuous controller N is received, in hertz. Examples: amplfo_freqcc1=5 amplfo_freqcc1=-12	floating point	0 hertz	-200 to 200 hertz
amplfo_freqchanaft	Amplifier LFO frequency change when channel aftertouch MIDI messages are received, in hertz. Examples: amplfo_freqchanaft=10 amplfo_freqchanaft=-40	floating point	0 hertz	-200 to 200 hertz
amplfo_freqpolyaft	Amplifier LFO frequency change when polyphonic aftertouch MIDI messages are received, in hertz. Examples: amplfo_freqpolyaft=10 amplfo_freqpolyaft=-4	floating point	0 hertz	-200 to 200 hertz
Equalizer				
eq1_freq **eq2_freq** **eq3_freq**	Frequency of the equalizer band, in Hertz. Examples: eq1_freq=80 eq2_freq=1000 eq3_freq=4500	floating point	eq1_freq=50 eq2_freq=500 eq3_freq=5000	0 to 30000 Hz
eq1_freqccN **eq2_freqccN** **eq3_freqccN**	Frequency change of the equalizer band when MIDI continuous control N messages are received, in Hertz. Examples: eq1_freqcc1=80	floating point	0	-30000 to 30000 Hz
eq1_vel2freq **eq2_vel2freq** **eq3_vel2freq**	Frequency change of the equalizer band with MIDI velocity, in Hertz. Examples: eq1_vel2freq=1000	floating point	0	-30000 to 30000 Hz
eq1_bw **eq2_bw** **eq3_bw**	Bandwidth of the equalizer band, in octaves. Examples: eq1_bw=1 eq2_bw=0.4 eq3_bw=1.4	floating point	1 octave	0.001 to 4 octaves
eq1_bwccN **eq2_bwccN** **eq3_bwccN**	Bandwidth change of the equalizer band when MIDI continuous control N messages are received, in octaves. Examples: eq1_bwcc29=1.3	floating point	0	-4 to 4 octaves
eq1_gain **eq2_gain** **eq3_gain**	Gain of the equalizer band, in decibels. Examples: eq1_gain=-3 eq2_gain=6 eq3_gain=-6	floating point	0 dB	-96 to 24 dB
eq1_gainccN **eq2_gainccN**	Gain change of the equalizer band when MIDI continuous control N messages are received, in decibels.	floating point	0 dB	-96 to 24 dB

eq3_gainccN	Examples: eq1_gaincc23=-12			
eq1_vel2gain **eq2_vel2gain** **eq3_vel2gain**	Gain change of the equalizer band with MIDI velocity, in decibels. Examples: eq1_vel2gain=12	floating point	0	-96 to 24 dB
Effects				
effect1	Level of effect1 send, in percentage (reverb in sfz). Examples: effect1=100	floating point	0	0 to 100 %
effect2	Level of effect2 send, in percentage (chorus in sfz). Examples: effect2=100	floating point	0	0 to 100 %

Examples

Example .sfz definition files showing every opcode functionality can be found here.

Version: 1.02, *Last updated on:* 10/1/2010