TeAr

The Text Arpeggiator

Pattern language reference

FX-Mechanics

Syntax version 2 — TeAr 0.4.0

Abstract

A TeAr pattern is a short string of text that turns the notes you hold into a sequence. Each pattern belongs to one arpeggiator, and several arpeggiators run at once, each with its own pattern, step duration and MIDI output channel.

This document is the complete reference for the pattern language as implemented in TeAr 0.4.0. Section 12 is a one-page summary for printing; the rest explains each command in turn.

Contents

1 How a pattern is read

A pattern is read left to right, one *step* at a time. At each step of the arpeggiator's clock the engine consumes characters until it finds a *note command*, plays whatever that command asks for, and stops there until the next step. When the end of the string is reached it wraps around to the beginning.

Characters fall into three groups:

Note commands
: End a step and produce a note (or a deliberate silence). Each one costs exactly one step of the clock.

Prefixes
: Modify the note that follows: pitch, velocity, octave, probability. They cost no time of their own.

Block markers
: Group steps together to scope a modifier or to change how degrees are interpreted. They cost no time either.

Whitespace is free. Spaces are ignored entirely, so `1 2 3`, `123` and `1  2  3` are the same pattern. Use spacing to make a pattern readable; it never changes what you hear.

Degrees, not pitches. A pattern never names an absolute pitch. It names a *degree* of whatever you are holding, so the same pattern follows your chords. How degrees map to pitch depends on the chord method:

Chord method	**Meaning of degree** n
Notes played	The nth note of the chord you are holding.
Chord played as is	The nth note, kept at the pitch you played it.
Single note	The nth degree of the selected scale, relative to the note you press.

If a degree is not present in the chord, the engine substitutes a neighbouring one rather than falling silent. Degrees beyond the size of the chord wrap around, so degree 5 of a three-note chord is degree 2.

2 Values

Degrees and velocity levels are written as a **single uppercase hexadecimal character**:

`0 1 2 3 4 5 6 7 8 9 A B C D E F` = 0 to 15

One character per value keeps patterns compact and keeps every step the same width on screen. Uppercase only, because `b` is the flat command (section 4) and a lowercase hexadecimal `b` could not be told apart from it.

Octave and probability arguments stay **decimal** (`0`–`9` and `0`–`7` respectively); their useful ranges fit in one decimal digit, so there is nothing to gain from extending them.

3 Note commands

Each of these ends a step. Everything else in the language is a prefix or a marker attached to one of them.

Command	Effect
`1`–`F`	Play that degree (1 = fundamental, 2 = second, and so on up to 15).
`0`	Play the current degree without naming one. Used to close a step that only carries modifiers, as in `vF0`.
`.`	Rest. Nothing sounds, and the step still takes its time.
`_`	Sustain. The previous note keeps ringing through this step.
`+`	Play the next degree after the last one played.
`-`	Play the previous degree.
`=`	Repeat the last degree played.
`?`	Play a random degree that is present in the chord.

Rest versus sustain. `.` cuts the note; `_` lets it ring. `1 . . .` is a short note followed by silence, `1 _ _ _` is one note four steps long.

4 Pitch modifiers

Prefixed to a note command. They affect only that one note.

Command	Effect
`#`	Raise the next note one semitone. Example: `#3`
`b`	Lower the next note one semitone. Example: `b3`

Note that `b` in lowercase is always the flat command, while uppercase `B` is the degree 11. This is the reason the value alphabet in section 2 is uppercase.

5 Velocity

Velocity has 15 levels, written in hexadecimal, spread evenly over the MIDI range: level n produces round($n \times 127/15$). So `1` is 8, `8` is 68, and `F` is 127.

Where a pattern sets no velocity, velocity follows your playing: each note is sounded at the velocity of the note you are holding, rounded to the nearest of the 15 levels. Until a note has been played the arpeggiator uses 96.

A `V` overrides the played velocity for as long as it is in scope. The two are tracked separately, so playing a new note does not disturb a `V`, and a `V` written inside a `( )` block stops applying at the closing bracket: after it, notes follow your playing again unless another `V` outside the block was already in force.

Lowercase `v` is **local**: it applies to the next note only. Uppercase `V` is **global**: it applies from that point on, until another `V` changes it or an enclosing block ends (section 8).

Command	Effect
v*N*	Set the next note's level to *N* (1–F).
v+	One level louder, for the next note only.
v-	One level quieter, for the next note only.
v?	A random level, for the next note only.
V*N*	Set the global level to *N*.
V+	One level louder, from here on.
V-	One level quieter, from here on.
V?	A random global level.

The relative forms step from whatever level is in effect at that point, and they *saturate* rather than wrap: V+ at level F stays at F, and V- floors at level 1, so a run of them can never silence a pattern.

Beware of unbalanced global steps. A pattern loops, so a V- with no matching V+ applies again on every pass and walks the velocity down until it reaches the floor. Either balance them, or scope them inside a block (section 8), or use the local v forms.

```
V8 vF1 2 3 vB2 3 vD1 2 3
```

sets a moderate base level, then accents individual steps without any drift.

6 Octave

Same shape as velocity: lowercase o is local to the next note, uppercase O is global. Arguments are decimal, 0–7.

Command	Effect
o*N*	Play the next note in octave *N*.
o+	One octave up, next note only.
o-	One octave down, next note only.
o?	A random octave, next note only.
O*N*	Set the global octave to *N*.
O+	One octave up, from here on.
O-	One octave down, from here on.
O?	A random global octave.

The relative forms clamp to the range 0–7. The default octave is 4.

Prefixes stack. Because octave commands are prefixes, o-o- means "down an octave, then down again". To drop an octave and then play the previous degree, write o-: the first two characters are the octave command, the third is the note command.

The same drift warning as for velocity applies: an unbalanced global O- lowers the octave on every pass of the loop. Use o- or a block.

7 Probability

p*N* makes the following step happen with probability $N \times 10\%$, where *N* is a decimal digit 0–9. p0 never fires and p9 fires nine times out of ten.

What happens on a failed roll depends on whether a fallback is given after a colon. Without one, the step becomes a rest.

Form	Behaviour
`p5 1`	Degree 1 half the time, otherwise a rest.
`p3 1:2`	Degree 1 three times in ten, otherwise degree 2.
`p7 1:.`	Degree 1 seven times in ten, otherwise an explicit rest.
`p5 1:_`	Degree 1 half the time, otherwise sustain the previous note.
`p5 1:?`	Degree 1 half the time, otherwise a random chord note.
`p5 (1 2 3)`	The whole group half the time, otherwise silence.
`p5 (1 2 3):(4 5)`	Degrees 1–3 on success, degrees 4–5 on failure.
`p8 (O+ 1 2 3)`	The group an octave up, eight times in ten.

Groups of unequal length. When the two branches of a `:` contain different numbers of steps, the loop's length depends on the roll, and so does where the pattern sits against the bar. This is legal and musically useful, but it means the sequence is no longer a fixed number of steps. Patterns that must stay locked to the bar should keep both branches the same length.

8 Blocks

8.1 Modifier scope: `( ... )`

Any global modifier used inside parentheses applies only within them. At the closing bracket the global octave and velocity are restored to the values they had at the opening bracket. For velocity that includes the case where no `V` was in force outside the bracket: the notes after it go back to following your playing.

```
1 2 (O+ 1 2 3) 1 2
```

Steps 3–5 sound an octave up; steps 1, 2, 6 and 7 are unaffected. Because the octave is restored, this pattern can loop forever without drifting, which a bare `O+` could not.

Parentheses nest.

8.2 Root-relative notes: `" ... "`

In *Single note* (scale) mode, degrees are normally read relative to the degree of the note you press. Steps between double quotes are instead read relative to the **scale root**, as though the root were being held, whatever you actually play.

```
1 2 "1 2 3" 4
```

Steps 3–5 stay anchored to the scale root while the rest follow the pressed note. This is how you write a fixed counter-melody under a moving line.

The quote characters are markers, not steps: the pattern above is seven steps long, not nine.

9 Counting steps

Exactly one thing costs time: a note command. Prefixes (`#`, `b`, `v`, `V`, `o`, `O`, `p`) and block markers (`(`, `)`, `"`) do not, and neither does whitespace.

Pattern	Steps	Why
`1 2 3`	3	three note commands
`vF1 2 3`	3	`vF` is a prefix on the first
`1 b2`	2	`b` prefixes the `2`
`1 B 2`	3	uppercase `B` is degree 11
`1 2 (0+ 1 2 3) 1 2`	7	brackets are markers
`1 "2 3" 4`	4	quotes are markers
`p5 1:2`	1	one step, two possible outcomes

The step duration itself is not part of the pattern: it is the arpeggiator's own setting, chosen from `1/4` down to `1/64`, including triplets. Two arpeggiators running the same pattern at different step durations is the simplest way to build a polyrhythm.

10 Worked examples

Pattern	What it does
`1 2 3 4`	A plain ascending arpeggio through the first four degrees.
`1 2 3 (0+ 1 2 3) 3 2 1`	Up, up an octave, then back down. The bracket restores the octave, so it loops cleanly.
`1 . 3 .` and `. 2 . 4`	Two arpeggiators interlocking: the second fills the gaps of the first. Give them different MIDI channels to split them across instruments.
`1 . . 2 . . 3 .`	A three-against-eight Euclidean feel, with the degrees walking up.
`p9 1 p5 3:. p7 2 p4 1:_`	A part that never repeats exactly: each step has its own likelihood, and two of them have fallbacks.
`V8 vF1 2 3 vB2 3`	A moderate base velocity with accents on the first and fourth steps.
`1 2 3 4 5 "1 3 5" 4 2`	In scale mode, a running line that drops to a fixed root-anchored figure in the middle.

11 Changes in syntax version 2

TeAr 0.4.0 introduced version 2 of the pattern syntax.

- Degrees and velocity levels became uppercase hexadecimal. Degrees now reach 15, so chords and scales with more than nine notes are fully addressable, and velocity has 15 levels instead of 8.
- `v+`, `v-`, `V+` and `V-` now work. They appeared in earlier documentation but had no effect.

Only velocity changed meaning. Patterns saved by earlier versions are converted automatically when a session or a preset is loaded, so they keep the velocities they had.

The conversion runs only where TeAr controls the load path. A pattern you retype or paste in by hand is read with the new meaning, so an old `v8` now means roughly half velocity rather than maximum; write `vF` instead.

12 Reference

Command	Scope	Effect
Note commands *each one takes exactly one step*		
`1`–`F`	—	Play that degree, 1 (fundamental) to 15.
`0`	—	Play the current degree without naming one, as in `vF0`.
`.`	—	Rest: nothing sounds, the step still takes its time.
`_`	—	Sustain: the previous note rings on through this step.
`+`	—	The next degree after the last one played.
`-`	—	The previous degree.
`=`	—	Repeat the last degree played.
`?`	—	A random degree present in the chord.
Pitch *prefix, affects the next note only*		
`#`	local	Raise the next note one semitone.
`b`	local	Lower the next note one semitone. Uppercase `B` is degree 11.
Velocity *levels 1–F over 1–127; follows the played note by default*		
v*N*	local	Set the next note's level.
`v+` `v-`	local	One level louder / quieter, next note only.
`v?`	local	A random level, next note only.
V*N*	global	Set the level from here on.
`V+` `V-`	global	One level louder / quieter, from here on.
`V?`	global	A random level from here on.
Octave *decimal 0–7; default 4*		
o*N*	local	Play the next note in that octave.
`o+` `o-`	local	One octave up / down, next note only.
`o?`	local	A random octave, next note only.
O*N*	global	Set the octave from here on.
`O+` `O-`	global	One octave up / down, from here on.
`O?`	global	A random octave from here on.
Probability *decimal 0–9, in tenths*		
p*N* *x*	—	Play *x* with probability $N \times 10\%$, else rest.
p*N* *x*`:`*y*	—	Play *y* instead when the roll fails.
p*N* `(...)`	—	Either branch may be a group of steps: p*N* `(...):(...)`.
Blocks *markers, no step cost*		
`( ... )`	—	Scope global modifiers; octave and velocity are restored at the closing bracket.
`" ... "`	—	Read degrees from the scale root instead of the pressed note (scale mode).

Things that catch people out	
Spaces are free	`1 2 3` and `123` are the same pattern.
`b` versus `B`	Lowercase is flat, uppercase is degree 11.
Globals drift	An unbalanced `V-` or `O-` re-applies on every loop.
Only notes cost time	Prefixes and block markers take no step.