

Spec reference

The input spec is YAML. **Everything is optional** — omitted fields are sampled from typologically-plausible defaults, so a two-line spec is a valid spec.

```
name: Veshtari
seed: 42

phonology:
  consonants: [p, t, k, m, n, s, l, r, v, sh, ch]
  vowels: [a, e, i, o, u]

syllables: [CV, CVC, V]    # C = consonant slot, V = vowel slot
word_count: 150

lexicon:                    # pin specific words; they are locked by default
  - gloss: water
    root: vela
```

Keys

key	meaning
name	the language id (slugged into the model id).
seed	integer driving all sampling; same spec + seed ⇒ identical bytes.
phonology.consonants, phonology.vowels	the inventory symbols.
phonology.ipa	treat the inventory as IPA — see IPA inventories .
phonology.orthography	symbol → spelling overrides (both modes).
syllables	syllable templates over C / V, e.g. [CV, CVC, V] .
spec_version	required "2.0" .

key	meaning
<code>concepts</code>	explicit <code>{gloss, pos}</code> list instead of the built-in list.
<code>word_count</code>	target lexicon size (trims or extends the concept list).
<code>lexicon</code>	pinned words (<code>gloss</code> , <code>root</code> , <code>pos</code> , <code>inflection_class</code> , <code>locked</code> , optional <code>field</code> / <code>senses</code> — see Polysemy).
<code>affixes</code>	pinned affixes (<code>id</code> , <code>gloss</code> , <code>form</code> , <code>position</code> , <code>applies_to</code> , <code>allomorphs</code> , <code>categories</code> , <code>inflection_class</code> , <code>locked</code>).
<code>category_order</code>	the morphology slot order.
<code>constraints.onsets</code> , <code>constraints.codas</code>	allowed onset / coda sequences (see Onsets and codas).
<code>constraints.forbidden</code>	forbidden sequences.
<code>semantics.domain</code> , <code>semantics.extensions</code>	the declared interpretation (<code>pos:gloss</code> keys will resolve to lexemes).
<code>syntax.word_order</code>	<code>SOV</code> / <code>SVO</code> / <code>VSO</code> .
<code>syntax.alignment</code>	<code>accusative</code> / <code>ergative</code> .
<code>discourse.particles</code>	opt in to the discourse-particle class (<code>dsc</code>).
<code>discourse.particle_position</code>	<code>initial</code> / <code>final</code> / <code>second</code> ; unset follows the subordinator edge.
<code>discourse.switch_reference</code>	opt in to <code>SS</code> / <code>DS</code> clause affixes on dependent verbs.
<code>discourse.reflexives</code>	opt in to the <code>refl</code> / <code>recip</code> anaphor set (<i>self, each other</i>).
<code>discourse.evidentiality</code>	opt in to the <code>evi</code> particles (<code>VIS</code> / <code>SENS</code> / <code>INFER</code> / <code>REP</code>).
<code>lineage</code>	<code>parent</code> + the ordered sound-change <code>rules</code> (set by <code>evolve</code>).

An explicit, **unsized** `lexicon` is treated as the complete word list (that is what `accept` writes back); otherwise the built-in concept list is used and `word_count` trims or extends it.

Polysemy

A lexeme can carry **added senses** — extra meanings motivated from its primary sense (ADR 0019). On a lexicon entry, `field` is the primary sense's **semantic field**, and `senses` is a list of added senses, each with its own `gloss`, `field`, a `relation` to the primary, and an optional free-text `rationale`.

```
lexicon:
  - gloss: crane
    root: kran
    pos: noun
    field: birds                # the primary sense's field (the donor)
    senses:
      - gloss: lifting machine
        field: tools           # the target field
        relation: METAPHOR     # METAPHOR | METONYMY | GENERALIZATION | SPECIALIZATION
        rationale: the jib looks like the bird's neck
```

A sense **shares its lexeme's part of speech and places** — it carries neither, so a cross-POS extension (noun → verb) is out of scope. Senses are **author data only**: there is no generator path for them, so `grow`-added and sampled glosses never gain one. `field` and `relation` are closed sets validated at load, and the rendered grammar sketch shows the senses in a `## Polysemy` section. See `spec/example-polysemy.yaml` for a worked example.

IPA inventories

By default a symbol *is* its spelling (`sh` is written `sh`). Set `phonology.ipa: true` to declare the inventory phonemic: each symbol is an IPA phoneme, and the tool derives a romanised **orthography** for the dictionary.

```
phonology:
  ipa: true
  consonants: [p, t, k, ʃ, tʃ, m, n, s, l]
  vowels: [a, e, i, o, u]
  orthography:      # optional; corrects or extends the built-in defaults
    tʃ: ch
    ʃ: sh
```

`/ʃ/` spells `sh` and `/tʃ/` spells `ch` from a built-in table; anything absent spells itself, so a plain Latin inventory is unchanged. `orthography` overrides the table (set an entry back to its own symbol to cancel a default), and the resolved map is stored on the inventory. The dictionary and word-form tables then show a `spelling` column beside the `ipa` one, and `accept` writes the map back.

Round-tripping (`accept`)

`accept` materialises a generated model into a fully-explicit spec. Regenerating from the result reproduces the language exactly:

```
conlang generate --spec spec/example.yaml --out out/veshtari.json
conlang accept   --model out/veshtari.json --out spec/veshtari.yaml
conlang generate --spec spec/veshtari.yaml --out out/veshtari.json
```

A few fields legitimately change across the round trip: `origin` becomes `specified` and `locked` becomes `true` for every accepted entry (pinned now, not sampled), and `spec_hash` changes because the spec text is different. Everything linguistic is identical, and `tests/test_invariants.py` asserts exactly that.

Onsets and codas

If your `syllables` include `CCV / CCVC`, the generator samples a legal onset set and **builds roots from it**, so generated words are legal by construction rather than by luck. `constraints.onsets` in the model records what was sampled; an explicit `constraints.onsets` in the spec always wins. See the [feature guide](#) for the legality rules.