

Bovnar — UCUM Unit Profile

Bovnar (BVNR) v1.1 documentation · UCUM Unit Profile · 2026-07-27

Spec version: 1.1 — the notation described here is UNDER IMPLEMENTATION.

It is not part of any published specification, and the version it will ship under is not settled (§2.2) **Status:** Under implementation — the code is in `src/utils/bovnar_ucum.c` and pinned by `tests/bovnar_ucum_test.c`, but nothing here is released: no published specification defines the notation, and `bovnar version` reports spec 1.1. Section 10.4 lists the parts of this document that were not built at all. **Scope:** How a UCUM expression may be written in the unit slot beside Bovnar's native notation, what it translates to, what it refuses, and what the format still guarantees once a foreign vocabulary is admitted.

Companion to [Unit & Currency Reference](#) (the native registry and notation grammar this profile sits beside), [Unit Ambiguities](#) (how a unit token is resolved, and the pairs that look interchangeable), and `doc/parser_level_unit_policy.md` (the reader- and writer-side unit policies a profile unit has to survive unchanged — a working note, not part of the published documentation set).

Every acceptance, refusal and conversion factor quoted below was produced by running the reference implementation built from this tree, and the behavioural claims are pinned by `tests/bovnar_ucum_test.c` (105 assertions). What is **not** machine-verified, and must not be read as such, is the UCUM side: the transliteration table in `src/gendata/ucum.bvnr` states what each UCUM atom is worth, and nothing in this repository checks that against UCUM's own publication. §9.2 says exactly what the generator does and does not prove, and §10.2 treats the gap as the standing risk it is.

Table of Contents

1. Overview

- 1.1 What a profile is
- 1.2 Why a notation rather than more native units
- 1.3 The one thing that must not change

2. Syntax

- 2.1 The namespace discriminator
- 2.2 Where a profile unit may appear, and in which documents
- 2.3 Five bytes the lexer has to learn
- 2.4 Commas inside an annotation

- 2.5 Length budget
- 2.6 Grammar
- 3. Translation
 - 3.1 Three outcomes, and no fourth
 - 3.2 Atoms and prefixes
 - 3.3 Operators, exponents, grouping
 - 3.4 Annotations
 - 3.5 Scale factors and the decade fold
 - 3.6 Arbitrary units
 - 3.7 Special units
 - 3.8 Affine units
- 4. Semantics after translation
 - 4.1 Equality
 - 4.2 Compatibility and conversion
 - 4.3 Policy, normalisation, and the writer
 - 4.4 Invariants the profile preserves
- 5. Serialisation
 - 5.1 Canonical output
 - 5.2 What canonical output loses, and on which path
 - 5.3 Emitting UCUM from a native unit
- 6. The transliteration table
 - 6.1 Verified mappings
 - 6.2 Collisions — the same spelling, a different unit
 - 6.3 Traps that are not spelling collisions
 - 6.4 Codes with no Bovnar representation
- 7. Data model
 - 7.1 The arbitrary-unit block
 - 7.2 Incommensurability, via the mechanism currencies already use
 - 7.3 No new field on the data event
 - 7.4 New error codes
- 8. API
 - 8.1 C
 - 8.2 Python
 - 8.3 CLI
- 9. Build and conformance
 - 9.1 Where the table lives
 - 9.2 What the generator checks, and what it does not
 - 9.3 Tests
 - 9.4 No build switch

10. Cost, risk, and what is left out

- 10.1 What it cost
- 10.2 What can go wrong
- 10.3 Deliberately not attempted
- 10.4 Specified here but not built
- See also

1. Overview

1.1 What a profile is

A **unit profile** is an alternative spelling for the unit slot of a type annotation. It is not a second unit model. A profile expression is translated, at parse time, into exactly the same `value_unit_t` a native expression produces, and from that point on nothing downstream can tell which notation the document used:

```
#!/bovnar 1.2
.systolic_a = <float_dec:64,ucum:mm[Hg]> 120.00; # profile spelling
.systolic_b = <float_dec:64,mmHg> 120.00; # native spelling – same value_unit_t
```

`bvn_unit_equal` reports those two units equal. `bvn_units_compatible`, `bvn_unit_convert_factor`, the reader and writer unit policies, `bvnr_normalise_si`, the DOM and the `want_unit` hook all keep working with no knowledge that a profile exists. That is the whole design constraint, and §3.1 exists to hold it: a profile expression either becomes a real `value_unit_t` or it becomes an error. There is no third state in which a value carries a unit the rest of the library cannot reason about.

The general form is `namespace:code`. This document defines exactly one namespace, `ucum`. The grammar is written for more (§2.6) so that a later `cf:` or `udunits:` costs no new syntax, but nothing else is specified here and an unknown namespace is an error, not a passthrough.

1.2 Why a notation rather than more native units

Bovnar's native registry is 180 physical units and 216 currencies, hand-maintained in `src/gendata/`. UCUM's atom table is larger — a complete clinical, apothecary, troy, avoirdupois and CGS inventory — and its expression language is unbounded, so the set of valid UCUM codes cannot be enumerated as a table of units at all.

Absorbing that into the native registry would mean adding several hundred aliases, most of them spellings no Bovnar document would ever choose, and several of which collide with existing native symbols (§6.2). A notation costs one table of mappings instead, keeps

the two disambiguation regimes apart, and lets a document produced by a UCUM-speaking system be read without the native registry growing at all.

1.3 The one thing that must not change

The obvious way to get this wrong is to make `ucum:` a hole through which any string reaches a value uninspected. Then `<float:64,ucum:metre>` parses, the typo survives to the consumer, and a unit has arrived that the library cannot reason about — no dimension, no compatibility check, no conversion. Every guarantee the unit system provides is a guarantee about units it understands.

So the rule in §3.1 is absolute, and every later section is written to keep it: a `ucum:` expression is parsed as UCUM in full, against UCUM's own atom table, and anything that is not a valid UCUM expression over known atoms is `error_unit_illegal` before translation is even attempted. Passthrough exists (§3.6) but reaches only atoms UCUM itself defines and classifies as incommensurable. It is never a fallback for something unrecognised.

2. Syntax

2.1 The namespace discriminator

A profile unit is a namespace name, an ASCII colon, and a code:

```
ucum:mm[Hg]
ucum:10*3/uL
ucum:mL{total}
```

The colon is already an accepted byte inside a type-annotation body (state table `copy_type_byte`, `[0x3a]`), and no native unit alias or currency code contains one, so the discriminator is unambiguous against the entire existing registry with no lookahead.

It is also already an error today, which is what makes the extension safe. Against the shipped 1.1 build:

```
$ bovnar validate t.bvnr      # .a = <float:64,ucum:mmHg> 1.0;
Validation failed: unit_illegal at line 2, col 25
```

The lexer accepts the bytes and `bvn_parse_unit` rejects the string. No document that parses today contains a `ucum:` unit, so no document can change meaning when one starts to parse. Contrast the alternative of a quoted form (`ucum:"mm[Hg]"`), which would put a string-escape sub-language inside a type body for no gain.

2.2 Where a profile unit may appear, and in which documents

The notation is gated on the declared spec version. A profile unit needs a `#!bovnar` directive declaring a version above 1.1 — spelled `#!bovnar 1.2` today — exactly as the `datetime` family and the `\x/\u` escapes need a declared 1.1. That version is **not one this build advertises**: the notation is under implementation, `bovnar version` reports spec 1.1, and the number it finally ships under is not settled. Without one it is `error_unit_illegal` — in a 1.1 document `ucum:mm[Hg]` is simply not a unit, the same way `<datetime:64>` is simply not a value type in a 1.0 document. A document with **no** directive declares nothing and therefore gets neither surface.

```
#!bovnar 1.2
.systolic = <float_dec:64,ucum:mm[Hg]> 120.00;    # OK
```

Without the gate a document could carry a unit that every conforming reader of its own declared version must reject, which is the interoperability hazard the directive exists to prevent. Native units are unaffected in every version: the bump is additive, and `<float:64,mmHg>` parses under 1.0 as it always did.

The writer enforces the other half. A unit with no native spelling — one carrying a UCUM arbitrary atom — can only be emitted in this notation, so writing one without having emitted the opt-in directive is `error_unsupported_spec_version` rather than a document the library cannot read back. A translated unit needs no such guard: `ucum:mm[Hg]` is written as the native `mmHg`, which every version accepts.

Otherwise: everywhere a native unit may appear — as the unit parameter of a type annotation, and as an inline unit suffix. Parameter ordering stays free (doc/2 §2.1) and the annotation/inline agreement rule (doc/2 §2.2) is unchanged — the comparison is on the parsed `value_unit_t`, so the two spellings may differ as long as they mean the same thing:

```
#!bovnar 1.2
.a = <float:64,ucum:mm[Hg]> 120.0;
.b = 120.0 ucum:mm[Hg];
.c = <float:64,mmHg> 120.0 ucum:mm[Hg];    # OK — both parse to the same unit
.d = <float:64,m> 1.0 ucum:s;               # error_unit_mismatch, as always
```

A profile unit is a unit, so it is confined to the same type families (doc/2 §2.3): `uint`, `sint`, `float`, `float_fix`, `float_dec`. A `ucum:` parameter on `utf8`, `bool` or `datetime` is `error_illegal_value_type`, unchanged.

Currencies stay native-only. UCUM has no monetary codes, `ucum:` never yields one, and the `$` sigil rule (doc/2 §9.1) is untouched.

2.3 Five bytes the lexer has to learn

UCUM needs `[`, `]`, `{`, `}` and `'` — the last for codes like `[arb'U]` and `[Amb'a'1'U]`. All five were rejected by the type-body and inline-unit byte classes:

```
$ bovnar validate t.bvnr          # .a = <float:64,ucum:mm[Hg]> 1.0;
Validation failed: unexpected_input_byte at line 2, col 18
```

The change is four table entries in each of two states of `src/lexer/bovnar_state_table.c`:

Byte	Char	States extended
0x27	'	<code>copy_type_byte</code> , <code>type_body_outro</code> , <code>inline_unit_body</code>
0x5B	[	<code>copy_type_byte</code> , <code>type_body_outro</code> , <code>inline_unit_body</code>
0x5D	]	<code>copy_type_byte</code> , <code>type_body_outro</code> , <code>inline_unit_body</code>
0x7B	{	<code>copy_type_byte</code> , <code>type_body_outro</code> , <code>inline_unit_body</code>
0x7D	}	<code>copy_type_byte</code> , <code>type_body_outro</code> , <code>inline_unit_body</code>

`type_body_outro` is extended alongside `copy_type_byte` because the two already carry an identical unit-byte class; letting them diverge here would be the odd choice, and anything nonsensical that reaches the type-spec parser through it is still `error_illegal_value_type`.

This is the only lexer change the profile requires, and it is the one place where the profile is not invisible to a native-only document: after it, `<float:64,m[s]>` reaches `bvn_parse_unit` and fails there (`error_unit_illegal`) instead of failing in the lexer (`error_unexpected_input_byte`). One error code moves for a family of inputs that were errors before and are errors after. §9.3 pins that in the conformance corpus rather than leaving it to be discovered.

Nothing else about the byte class changes. `<`, `>`, `"`, `;`, `#` and whitespace stay out, so a brace cannot swallow the rest of the document: an unclosed `{` runs to the value terminator and ends as `error_unit_illegal`, not as a parse that consumes the file.

That fences off one thing UCUM permits. UCUM allows any printable ASCII except braces inside an annotation; this profile allows only the bytes a type body accepts, because `;` and `#` end a value and `<` and `>` delimit a type annotation. `ucum:mL{a;b}` is `error_unit_illegal`. The alternative — admitting a byte that terminates the construct it sits inside — is not a trade worth making for an annotation the unit model ignores anyway.

2.4 Commas inside an annotation

A type-parameter list is comma-separated, and a UCUM annotation may contain a comma (`{cells,total}`). The parameter scanner in `bvn_parse_type_annotation` therefore tracks brace depth: a `,` at depth 0 ends the parameter, a `,` at depth ≥ 1 is part of it. Bracket depth needs no such treatment — `[ ... ]` cannot contain a comma in any UCUM atom — but is tracked anyway so that an unbalanced bracket is diagnosed as a malformed unit rather than as a malformed parameter list.

Depth is bounded by `BVN_UNIT_GROUP_MAX_DEPTH` (16), the same bound the native parentheses parser uses. UCUM annotations do not nest — `{` inside an annotation is not legal UCUM

— so any depth above 1 is already an error; the bound exists so that the scanner cannot be driven to recurse by a hostile document before the parser gets to say so.

2.5 Length budget

The unit parameter is capped at `UINT8_MAX` (255) bytes by the existing type buffer, and the cap is enforced before parsing, as `error_unit_too_long`. UCUM codes with long annotations can exceed it — `mL/min/{1.73_m2}` does not, but a genuinely chatty annotation will. No new limit and no new error: the profile inherits this one, and a producer that needs more than 255 bytes of unit is telling you the annotation is a field, not a unit.

2.6 Grammar

The formal rules live in [the EBNF](#) beside `unit-param`, which is where the byte classes of §2.3 are also recorded:

```
unit-param  = profile-unit | native-unit ;
profile-unit = profile-name , ":" , profile-code ;
profile-name = lower-alpha , {lower-alpha} ;      (* "ucum" *)
profile-code = profile-char , {profile-char} ;
```

`profile-code` is deliberately not given a grammar. The sub-grammar is **semantic**, as the native unit sub-grammar is (doc/2 §5.2): the lexer captures bytes, and `bvn_parse_unit` — which dispatches on the namespace — decides whether they are a UCUM expression. The normative grammar for what is inside is UCUM's own, and restating it here would create a second authority to keep in step with the first.

3. Translation

3.1 Three outcomes, and no fourth

Every `ucum:` expression ends in exactly one of three states.

Out-come	When	Result
Translated	Every atom maps onto the native registry, and the whole expression is representable	A normal <code>value_unit_t</code> ; indistinguishable from the native spelling
Profile-only	The expression is valid UCUM and every atom is known, but one or more atoms are UCUM arbitrary units (§3.6)	A <code>value_unit_t</code> over the arbitrary-unit block: comparable, never convertible
Refused	Anything else	An error code, at parse time, from the parser

Refusal splits three ways by cause, because a producer needs to know which of these happened:

- not a valid UCUM expression, or an atom UCUM does not define → `error_unit_illegal`;
- valid UCUM, known atoms, no Bovnar representation (a special unit §3.7, an unrepresentable scale §3.5, more than `BVNR_MAX_UNIT_COMPONENTS` components) → `error_unit_profile_unsupported`;
- the namespace before the colon is not a profile this build knows → `error_unit_profile_unknown`.

There is no "opaque string" outcome. A unit the library cannot reason about is a unit that has escaped the enforcement point, and the format's only distinguishing claim is that units do not do that.

3.2 Atoms and prefixes

UCUM separates prefix from atom by its own rule; Bovnar separates it by longest-alias-suffix match with an optional explicit `~` (doc/2 §4.3). Translation happens **after** UCUM's split, on the resolved (prefix, atom) pair, never by handing the raw UCUM string to the native parser. That is what keeps the two disambiguation regimes from contaminating each other, and it is why the collisions in §6.2 are harmless rather than fatal.

A code is matched as a whole atom first, so a complete atom always outranks a prefixed reading of it — `mho` is the siemens, not milli-ho, and `cd` is the candela, not centi-day. Only if that fails is the code split, and then **every** prefix that heads it is tried, longest first. Trying only the longest is wrong: UCUM's prefixes are not suffix-free (`d` heads `da`), so `dar` — the deciare — matches `da`, leaves `r`, and would be refused although it is legal UCUM. A split whose atom is non-metric is not a match either, which is what keeps `k[arb'U]` an error while the walk continues to a shorter prefix.

A UCUM prefix becomes the corresponding `si_prefix_id_t` on the translated component. Two things can go wrong, and both are handled by the fold in §3.5 rather than by refusal:

- the native target does not accept prefixes at all (`prefix = none` or `ratio` in `src/gendata/units.bvnr` — `%`, `ppm`, `pH`, `PSU`, `mph`, the water-hardness degrees);
- the native target already carries a prefix because the mapping is not one-to-one (`kg` → `k~g`, `ar` → `c~ha`).

UCUM's binary prefixes (`Ki`, `Mi`, ...) map to `prefix_iec` and are subject to the same `bvn_prefix_unit_valid` check as natively — `Ki~B` is fine, `Ki~m` is not, in both notations.

3.3 Operators, exponents, grouping

UCUM	Bovnar	Note
<code>.</code>	<code>·</code>	multiplication

UCUM	Bovnar	Note
<code>/</code>	<code>/</code>	division — see below
<code>m2, s-1</code>	<code>m², s⁻¹</code>	exponent directly after the atom; range ± 9 as natively (doc/2 §6)
<code>()</code>	<code>()</code>	grouping, mapped through the native group parser (doc/2 §5.2)
<code>1</code>	(nothing)	the unity atom; contributes no component

The division rule is the one real difference and it must not be papered over. UCUM's `/` is a binary operator evaluated left to right: `a/b/c` is `(a/b)/c` — and so is Bovnar's, whose latching denominator gives `a·b-1·c-1`, the same thing. But UCUM's `/` applies only to the term that follows it, whereas Bovnar's latches for the rest of the expression. `a/b.c` is `a·c/b` in UCUM and `a·b-1·c-1` in Bovnar. Translation therefore works on UCUM's parse tree and emits explicit signed exponents, never a `/`-bearing native string:

```
ucum:kg.m/s2      → k~g·m·s-2
ucum:kg/m.s2      → k~g·m-1·s2      (NOT k~g/m·s2, which is kg·m-1·s-2)
ucum:/min         → min-1
```

A leading `/` is UCUM's reciprocal form and needs the unity atom to be dropped: `/min` has no numerator, and Bovnar has no way to write a bare `1` in a unit expression (`1` alone is `error_unit_illegal`), so the translation is the negative exponent, verified:

```
ucum:/min      → min-1      1 in coherent SI = 0.016666666666666666
ucum:/uL       → μ~L-1     1 in coherent SI = 999999999.9999999
```

3.4 Annotations

A UCUM annotation `{...}` carries no meaning; UCUM defines `{anything}` standing alone as the unity. The profile follows exactly, and states the consequence rather than hiding it:

```
ucum:mL{total}    → m~L      (equal to ucum:mL, and to native m~L)
ucum:{RBC}/uL     → μ~L-1
ucum:{cells}/uL   → μ~L-1   (equal to the line above)
ucum:{RBC}        → no_unit
```

`{RBC}/uL` and `{cells}/uL` compare equal because in UCUM they are equal. An annotation is a comment, not a discriminator. If the distinction between red cells and cells matters to the consumer, it is not a unit and does not belong in the unit slot — it belongs in a sibling field, where Bovnar can type it and the reader can find it by key path.

The annotation text is preserved for round-trip (§5.2), and is preserved verbatim, including case and spacing. It never affects equality, compatibility, conversion or normalisation.

This is the valve. UCUM's own justification for annotations is that a code system which refuses the expressiveness its users perceive gets adopted halfway, and halfway is as bad as not at all. Bovnar has no such valve natively and gains one here — bounded to a notation

where it is semantically inert, rather than added to the native grammar where it would have to mean something.

3.5 Scale factors and the decade fold

UCUM writes powers of ten as `10*n` or `10^n` and uses them heavily in clinical counts (`10*3/uL`). Bovnar has no component for a bare numeric factor. It does have prefixes, which are exactly a decimal decade attached to a component — so a scale factor is discharged into a prefix.

Let translation produce components $c_1 \dots c_n$ with exponents e_i and current prefix decades p_i , plus a residual decade D accumulated from every `10*n`, every `10^n`, and every prefix that its target refused to carry. Then:

Fold rule. Choose the leftmost i such that e_i divides D exactly, and $p'_i = p_i + D/e_i$ is a decade with a defined SI prefix, and `bvn_prefix_unit_valid` accepts that prefix on b_i . Set $p_i \leftarrow p'_i$ and $D \leftarrow 0$. If no such i exists, the expression is `error_unit_profile_unsupported`.

The division by e_i is what makes it correct in a denominator, and getting it wrong is a nine-orders-of-magnitude error rather than a cosmetic one. Verified against the implementation:

UCUM	Fold	Bovnar	1 unit in coherent SI
<code>10*3/uL</code>	$D = +3, e = -1, p = -6 \rightarrow p' = -9$	<code>n~L⁻¹</code>	<code>999999999999.9999</code>
<code>10*6/L</code>	$D = +6, e = -1, p = 0 \rightarrow p' = -6$	<code>μ~L⁻¹</code>	<code>999999999.9999999</code>
<code>10*9/L</code>	$D = +9, e = -1, p = 0 \rightarrow p' = -9$	<code>n~L⁻¹</code>	<code>999999999999.9999</code>
<code>10*12/L</code>	$D = +12, e = -1, p = 0 \rightarrow p' = -12$	<code>p~L⁻¹</code>	<code>999999999999999.9</code>
<code>10*3.m</code>	$D = +3, e = +1, p = 0 \rightarrow p' = +3$	<code>k~m</code>	<code>1000.0</code>
<code>10*-3.g</code>	$D = -3, e = +1, p = 0 \rightarrow p' = -3$	<code>m~g</code>	<code>1e-06</code>

(The last digits are the existing `bvni_ipow` rounding, not something the fold introduces; the exact path is `bvn_unit_convert_rational`, doc/2 §12.4.)

What the fold cannot do, and why that is stated rather than fixed. SI prefix decades are $\pm 1, \pm 2, \pm 3$, and then multiples of three to ± 30 . There is no prefix for $10^4, 10^5, 10^7$ or 10^8 , so `10*4/L` and `10*5.m` are `error_unit_profile_unsupported`. Two escapes were considered and both rejected: synthesising a dimensionless factor component out of `%` and `ppm` (`10*4 ≈ %-2`) abuses the ratio units and produces a unit no human would recognise; and adding a numeric-factor component to `value_unit_component_t` breaks the ABI for a case the clinical corpus does not appear to need. A named refusal is better than either.

3.6 Arbitrary units

UCUM's arbitrary units — `[IU]`, `U`, `[arb'U]`, `[PFU]`, `[CFU]`, and the rest of that table — are assay-defined. UCUM's own rule is that they are commensurable with nothing, including each other. They are also the single largest reason a real clinical code fails to map (§6.4).

The profile admits them, as a contiguous block of `value_base_unit_t` ids appended after the current last enumerator (§7.1), one id per UCUM arbitrary atom. Consequences, all of which fall out of the existing machinery:

- they are **profile-only**: no native alias, reachable only through `ucum:`, and serialised back as `ucum:[IU]` (§5.1). Nothing about the native notation changes;
- they are **mutually incommensurable**, by the range predicate of §7.2 rather than by one quantity kind each;
- they take prefixes if and only if UCUM says so (`[IU]` is metric in UCUM, so `k[IU]` translates; `[arb'U]` is not, so a prefix on it is `error_unit_illegal` — UCUM's error, raised before translation);
- they never appear in an SI normal form, never produce a conversion factor, and `bvnr_normalise_si` leaves them alone, exactly as it already leaves currencies and the dimensionless kinds alone.

The critical thing this is **not**: a single shared "opaque" base unit. Collapsing `[IU]` and `[PFU]` onto one id would make them compare equal, which is the silent-wrongness this format exists to refuse. One id each is the only honest encoding, and the ids come from UCUM's table rather than from anyone's judgement.

3.7 Special units

UCUM classes as special the units defined by a function rather than a factor: the temperature scales, the logarithmic ratios, the prism diopetre, the homeopathic potencies. Bovnar has native forms for some, and those translate (§6.1): `CeL`, `[degF]`, `[degRe]`, `[pH]`, `Np`, `B`, `deg`, `gon`.

The rest are `error_unit_profile_unsupported`. The bel-with-a-reference family is the instructive case: `B[SPL]`, `B[V]`, `B[W]`, `B[mV]`, `B[uV]` all differ from a plain bel only in their reference level, and Bovnar's `dB` carries no reference. Translating them to `da~dB` would discard exactly the information that distinguishes them and would let a sound-pressure level compare equal to a voltage level. Refusing is the whole point.

`B` → `da~dB` is admitted because 1 bel is 10 decibels, exactly, in the logarithmic domain, and `dB` carries its own quantity kind (`BVNI_KIND_LOG_DECIBEL`) so the result cannot drift into a plain count. Verified: `da~dB` has factor `10.0`. It is a deliberate call and the one place in the table where a prefix is used on a logarithmic unit.

3.8 Affine units

Nothing new. `Cel` translates to `°C`, and the native affine discipline (doc/2 §3, and the `.affine` / `.offset` fields in `src/gendata/units.bvnr`) applies unchanged: an affine unit is valid at exponent 1 only, and a compound containing one parses but yields no conversion value, because the offset is a kelvin count and a product with signature `K·s-1` has nowhere to put it.

`ucum:Cel/h` therefore behaves exactly as native `°C/h` does. The profile neither improves nor worsens this, and it does not close the temperature-**difference** gap — UCUM has no delta scale either, and writes a temperature difference as `K`. See §10.3.

4. Semantics after translation

4.1 Equality

`bvn_unit_equal` is unchanged: a multiset comparison of (base, exponent, prefix) triples. Because translation produces ordinary components, a profile unit and a native unit that mean the same thing compare equal, and the annotation/inline agreement check of doc/2 §2.2 works across notations:

```
#!/bovnr 1.2
.v = <float:64,mmHg> 120.0 ucum:mm[Hg];      # OK
.w = <float:64,ucum:kg.m/s2> 1.0 k~g·m·s-2;  # OK
```

Two profile units compare through their translations, so `ucum:mL{a}` equals `ucum:mL{b}` equals `ucum:mL`. §3.4 argues that is correct; §6.3 lists what it costs.

4.2 Compatibility and conversion

Unchanged, including the quantity-kind fences of `bvni_kinds_match`. A translated unit is subject to every one of them:

- `ucum:bit` and `ucum:By` do not convert into each other, because `b` and `B` are separate information kinds;
- `ucum:[pH]` does not convert into a plain count, because `pH` has its own logarithmic kind;
- `ucum:rad/s` does not convert into `ucum:Hz`, because the angle kind separates angular frequency from frequency;
- arbitrary units convert into nothing at all (§7.2).

4.3 Policy, normalisation, and the writer

`bvnr_unit_policy_t` takes its targets and rules as unit **strings**, parsed with `bvn_parse_unit` at the point the policy is set. Those strings therefore accept the profile notation for free:

```
static const bvnr_unit_rule_t rules[] = {
    { .path = ".patient.systolic", .unit = "ucum:mm[Hg]", .mode = bvnr_rule_require },
};
```

and the assertion is satisfied by a document that writes `mmHg`, or `ucum:mm[Hg]`, or `k~Pa` — because the rule is dimensional, evaluated on the parsed unit, and none of those three spellings survives into the comparison. Target lists, `bvnr_normalise_si`, `--require-dimension` and `--field` behave identically.

The writer's validation half (`bvnr_writer_set_unit_policy`) is likewise unchanged. A producer can be held to "every pressure is written in mm[Hg]" whether it spells that natively or not.

4.4 Invariants the profile preserves

All of the following hold for a profile unit exactly as they hold for a native one:

- the unit is inside the bytes, not in a sidecar attribute or an external table;
- it binds to one value, not to a variable or a file;
- the parser checks it, on every parse, with nothing for the application to call;
- a unit that does not parse stops the document — it does not arrive at the application as a string for the application to deal with;
- a unit that parses is dimensionally analysable. §3.1 is what buys this, and it is the reason there is no opaque-passthrough outcome.

5. Serialisation

5.1 Canonical output

`bvn_unit_to_string` emits the **native** canonical form whenever every component has one:

```
parse "ucum:mm[Hg]"    → write "mmHg"
parse "ucum:kg.m/s2"   → write "k~g·m·s-2"
parse "ucum:10*3/uL"   → write "n~L-1"
```

A unit containing a profile-only component (§3.6) has no native form, so the profile prefix becomes part of its canonical spelling and the whole expression is emitted in profile notation:

```
parse "ucum:[IU]/L" → write "ucum:[IU]/L"
```

Re-parsing either output yields the same `value_unit_t`, so round-trip is closed in both cases without any state outside the struct.

5.2 What canonical output loses, and on which path

Formatting a `value_unit_t` is lossy in exactly one respect: annotations. `bnv_unit_to_string` on `ucum:mL{total}` gives `m~L`, and `{total}` is gone; `ucum:{RBC}/uL` and `ucum:{cells}/uL` both give `μ~L-1`. Nothing else is lost — a translated unit's meaning survives exactly, and a profile-only unit round-trips byte-for-byte through §5.1.

Which path a document takes decides whether it sees that loss. Re-serialising a document the library parsed — `bovnr pretty-print`, the reader-to-writer round trip — re-emits the type annotation from the captured source text, so the producer's spelling survives verbatim, annotations included:

```
$ bovnr pretty-print ann.bvnr
.a = <float:64,ucum:mL{total}> 1.0;
.b = <float:64,_10,ucum:{RBC}/uL> 2.0;      # promoted from an inline unit
```

Constructing a document through the writer API is the path that loses it. `bnvnr_write_float` and its siblings take a `value_unit_t`, and a `value_unit_t` has nowhere to hold an annotation, so a producer that builds a document from parsed units writes the canonical form.

Making the writer preserve the spelling did not ship. It needs a field on `bnvnr_data_t` to carry the source out of the reader and a new parameter on every `bnvnr_write_*` entry point to carry it back in — a large change to the whole writer surface for a string the unit model ignores. See §10.4.

5.3 Emitting UCUM from a native unit

`bnv_unit_to_ucum(value_unit_t, char*, size_t)` writes a UCUM code for a unit that has one, for producers whose output field is UCUM-typed. It returns a negative length when it fails.

The mapping is driven by a **generated reverse table**, not by searching the forward one. Two things depend on that. First, the choice is deterministic and canonical: where several atoms name one base, the shortest wins, ties alphabetically — so siemens is `S` and not `mho`, and the calorie is `cal` and not `cal_th`. Searching the forward table picked whichever row its length-sorted order reached first, which got both of those wrong.

Second, each row carries the atom's own decade, which is what lets a base whose UCUM atom is itself prefixed be written at all. `m[Hg]` is a metre of mercury — bovnr's `mmHg` times 10^3 — so writing plain `mmHg` emits the prefix for `0 - 3` and produces `mm[Hg]`. The hectare is the same shape (`ar` carries `-2`, so `ha` becomes `har`). Without the decade those bases had no UCUM form at all.

It remains **partial by construction**: a native unit outside the transliteration table has nowhere to go. That is the Old German units, the water-hardness degrees, the turbidity kinds, `PSU`, `CF`, `mph`, `kph`, and every currency. The asymmetry is worth stating plainly: the profile is a good UCUM reader and a partial UCUM writer. A round trip that starts in UCUM returns to UCUM; one that starts in Bovnar's native registry may have nowhere to go.

Sweeping the whole native registry across twelve prefixes, 592 spellings survive a native → UCUM → native round trip unchanged and 1211 have no UCUM code; none round-trips to a different unit.

6. The transliteration table

6.1 Verified mappings

The shipped table is `src/gendata/ucum.bvnr`: 142 mapped **atoms**, 32 arbitrary units, 51 known but refused, and UCUM's 20 prefix spellings. What follows is a reading of it in terms of whole codes rather than atoms, which is how a producer meets it — `mg/dL` is three atoms and two prefixes, not a row.

Each Bovnar column below was produced by parsing that target with the reference implementation and reading back `bvn_unit_to_string` and the coherent-SI factor. The UCUM column is the table author's; §9.2 says what is and is not checked about it.

SI base and named derived

UCUM	Bovnar	1 UCUM unit in coherent SI
<code>m</code>	<code>m</code>	<code>1.0</code>
<code>s</code>	<code>s</code>	<code>1.0</code>
<code>g</code>	<code>g</code>	<code>0.001</code>
<code>kg</code>	<code>k~g</code>	<code>1.0</code>
<code>K</code>	<code>K</code>	<code>1.0</code>
<code>mol</code>	<code>mol</code>	<code>1.0</code>
<code>cd</code>	<code>cd</code>	<code>1.0</code>
<code>rad</code>	<code>rad</code>	<code>1.0</code>
<code>sr</code>	<code>sr</code>	<code>1.0</code>
<code>Hz</code>	<code>Hz</code>	<code>1.0</code>
<code>N</code>	<code>N</code>	<code>1.0</code>
<code>Pa</code>	<code>Pa</code>	<code>1.0</code>

UCUM	Bovnar	1 UCUM unit in coherent SI
J	J	1.0
W	W	1.0
V	V	1.0
Ohm	Ω	1.0
F	F	1.0
C	C	1.0
S	S	1.0
mho	S	1.0
Wb	Wb	1.0
T	T	1.0
H	H	1.0
lm	lm	1.0
lx	lx	1.0
Bq	Bq	1.0
Gy	Gy	1.0
Sv	Sv	1.0
kat	kat	1.0

Metric non-SI

UCUM	Bovnar	1 UCUM unit in coherent SI
L	L	0.001
l	L	0.001
t	t	1000.0
u	Da	1.6605390666e-27
eV	eV	1.602176634e-19
ar	c~ha	100.0
har	ha	10000.0
st	m ³	1.0
bar	bar	100000.0
atm	atm	101325.0
Ao	Å	1e-10
b	barn	1e-28
AU	au	149597870700.0

UCUM	Bovnar	1 UCUM unit in coherent SI
pc	pc	3.085677581491367e+16

Time

UCUM	Bovnar	1 UCUM unit in coherent SI
min	min	60.0
h	h	3600.0
d	d	86400.0
wk	wk	604800.0
mo_j	mo	2629800.0
a_j	yr	31557600.0
a	yr	31557600.0

Bovnar's `yr` is 31557600 s = 365.25 d and its `mo` is 2629800 s = 30.4375 d, which are the **Julian** year and month exactly — so `a_j`, `a` and `mo_j` map with no loss. The other UCUM year and month variants do not (§6.4).

Angle and logarithmic

UCUM	Bovnar	1 UCUM unit in coherent SI
deg	°	0.017453292519943295
gon	grad	0.015707963267948967
'	arcmin	0.0002908882086657216
''	arcsec	4.84813681109536e-06
circ	rev	6.283185307179586
Np	Np	1.0
B	da~dB	10.0
[pH]	pH	1.0

Temperature

UCUM	Bovnar	1 UCUM unit in coherent SI
Cel	°C	1.0
[degF]	°F	0.5555555555555556
[degRe]	°Re	1.25

Length, US and Imperial

UCUM	Bovnar	1 UCUM unit in coherent SI
[in_i]	in	0.0254
[ft_i]	ft	0.3048
[yd_i]	yd	0.9144
[mi_i]	mi	1609.344
[nmi_i]	nmi	1852.0
[ft_us]	ftUS	0.3048006096012192
[fth_i]	fath	1.8288
[fur_us]	fur	201.168
[ch_us]	ch	20.1168
[rd_us]	rd	5.0292
[acr_us]	ac	4046.8564224

Mass

UCUM	Bovnar	1 UCUM unit in coherent SI
[lb_av]	lb	0.45359237
[oz_av]	oz	0.028349523125
[dr_av]	dr	0.0017718451953125
[gr]	gr	6.479891e-05
[stone_av]	st	6.35029318
[oz_tr]	oz_t	0.0311034768
[pwt_tr]	dwt	0.00155517384
[sc_ap]	sc	0.0012959782
[car_m]	ct	0.0002
[ston_av]	tn_sh	907.18474
[lton_av]	tn_l	1016.0469088
[slug]	slug	14.593902937206364

Volume

UCUM	Bovnar	1 UCUM unit in coherent SI
[gal_us]	gal	0.003785411784
[qt_us]	qt	0.000946352946
[pt_us]	pt	0.000473176473
[foz_us]	fl_oz	2.95735295625e-05

UCUM	Bovnar	1 UCUM unit in coherent SI
[tbs_us]	tbsp	1.478676478125e-05
[tsp_us]	tsp	4.92892159375e-06
[gil_us]	gi	0.00011829411825
[fdr_us]	fl_dr	3.6966911953125e-06
[min_us]	minim	6.1611519921875e-08
[pk_us]	pk	0.00880976754172
[bu_us]	bsh	0.03523907016688
[bbl_us]	bbl	0.158987294928
[gal_br]	gal_uk	0.00454609
[qt_br]	qt_uk	0.0011365225
[pt_br]	pt_uk	0.00056826125
[foz_br]	fl_oz_uk	2.84130625e-05
[gil_br]	gi_uk	0.0001420653125

Pressure, energy, force, power

UCUM	Bovnar	1 UCUM unit in coherent SI
mm[Hg]	mmHg	133.322387415
[psi]	psi	6894.757293168362
att	at	98066.5
cal_th	cal	4.184
[Btu_IT]	Btu	1055.05585262
erg	erg	1e-07
[HP]	hp	745.6998715822702
[kn_i]	kn	0.5144444444444445
dyn	dyn	1e-05
gf	g·gn	0.00980665
kgf	kgf	9.80665
[lbf_av]	lbf	4.4482216152605
[g]	gn	9.80665

gf is the one row where a single UCUM atom becomes a two-component Bovnar expression: Bovnar has no gram-force, but **g·gn** is gram times standard gravity, dimensions `[1,1,-2,0,0,0,0]`, factor `0.00980665`. Nothing in the design requires a mapping to be atom-to-atom.

CGS and radiation

UCUM	Bovnar	1 UCUM unit in coherent SI
P	P	0.1
St	St	0.0001
Gal	Gal	0.01
G	G	0.0001
Mx	Mx	1e-08
Oe	Oe	79.57747154594767
sb	sb	10000.0
ph	ph	10000.0
Ci	Ci	37000000000.0
[RAD]	c~Gy	0.01
[REM]	rem	0.01
R	R	0.000258

Digital and ratio

UCUM	Bovnar	1 UCUM unit in coherent SI
bit	b	1.0
By	B	1.0
Bd	Bd	1.0
%	%	0.01
[ppth]	‰	0.001
[ppm]	ppm	1e-06
[ppb]	ppb	1e-09

Compound and clinical

UCUM	Bovnar	1 UCUM unit in coherent SI
m/s	m/s	1.0
m/s2	m/s ²	1.0
kg/m2	k~g/m ²	1.0
kg.m/s2	k~g·m/s ²	1.0
/min	min ⁻¹	0.016666666666666666
/uL	μ~L ⁻¹	999999999.9999999

UCUM	Bovnar	1 UCUM unit in coherent SI
10*3/uL	n~L ⁻¹	999999999999.9999
10*6/L	μ~L ⁻¹	999999999.9999999
mmol/L	m~mol/L	1.0
mg/dL	m~g/d~L	0.01
ng/mL	n~g/m~L	1.00000000000000002e-06
kat/L	kat/L	1000.0
eq	mol	1.0
meq/L	m~mol/L	1.0

6.2 Collisions — the same spelling, a different unit

These are the codes where the same bytes name different quantities in the two namespaces. They are harmless **only** because translation runs on UCUM's resolved atom, never by handing the UCUM string to `bvn_parse_unit` (§3.2) — every one of them would be a silent misreading if it did. Verified against the native parser:

Spelling	Native Bovnar	UCUM	Damage if confused
st	stone, 6.35029318 kg, dimension [0,1,0,...]	stere, m ³ , dimension [3,0,0,...]	mass read as volume
B	byte, information	bel, da~dB	a data size read as a level
b	bit, information	barn, 1e-28 m ²	a data size read as an area
Gb	gigabit (G~b , factor 1e9)	refused — b is non-metric in UCUM, so G + b is not a legal code	a prefixed reading that exists natively and not in the profile
a	not a unit	year (Julian)	— (Bovnar declines the ambiguity; see Unit Ambiguities)
AU	not a unit	astronomical unit	— (Bovnar spells it au)
gf	not a unit	gram-force	—
CeL	not a unit	degree Celsius	—

Every row above is asserted in `tests/bovnar_ucum_test.c`.

`st` is the dangerous one, because both sides parse and the dimensions differ. It is also the argument for the namespace being mandatory rather than a fallback: there is no reading of a bare `st` that could be made to serve both, and a profile that guessed would be wrong for one of the two populations every time.

6.3 Traps that are not spelling collisions

Three rows where the spellings differ but the meanings are close enough to be mapped wrongly by hand. Each is a factor error, not a syntax error, so nothing would catch it downstream.

eq is not val. Bovnar's `val` is documented in `src/gendata/units.bvnr` as the equivalent as used in water analysis, where the ions counted are divalent: its factor is `0.5`, so `m~val/L` is 0.5 mmol/L. UCUM's `eq` is the generic equivalent, defined as one mole. Mapping `eq` → `val` would be **wrong by exactly a factor of two** on every clinical electrolyte value. The table maps `eq` → `mol` and `meq/L` → `m~mol/L`, and leaves `val` alone as the water-analysis unit it is.

The calorie lines up and the BTU does not. Bovnar's `cal` is `4.184 J` — the thermochemical calorie, which is what UCUM's unqualified `cal` is, so `cal` and `cal_th` both map. Bovnar's `Btu` is `1055.05585262 J`, the **IT** BTU: only `[Btu_IT]` maps, and every other BTU variant — the unqualified `[Btu]` included — is refused as `error_unit_profile_unsupported`. Mapping `[Btu]` onto `Btu` would be wrong by about 0.07 %: small, dimensionally correct, and invisible to every later check, which is the worst shape an error can have here.

The apothecary dram is not the avoirdupois dram. `[dr_ap]` is 3.8879346 g and bovnar's `dr` is 1.7718 g — a factor of 2.2 apart. `[dr_av]` maps; `[dr_ap]` is refused. The apothecary ounce is the troy ounce and does map, which is exactly what makes the dram a trap.

The annotation is not a discriminator. `ucum:{RBC}/uL` and `ucum:{cells}/uL` compare equal (§3.4). That is UCUM's semantics faithfully implemented, and it is still a trap for anyone who expected the unit slot to carry the analyte.

6.4 Codes with no Bovnar representation

Class	Examples	Outcome
Arbitrary units	<code>[IU]</code> , <code>U</code> , <code>[arb'U]</code> , <code>[PFU]</code> , <code>[CFU]</code>	Profile-only (§3.6) — parsed, comparable, never convertible
Special units with a reference	<code>B[SPL]</code> , <code>B[V]</code> , <code>B[W]</code> , <code>B[mV]</code>	<code>error_unit_profile_unsupported</code> (§3.7)
Other special units	<code>[p'diop]</code> , <code>%[slope]</code> , <code>[hp'_X]</code>	<code>error_unit_profile_unsupported</code>
Scale outside a prefix decade	<code>10*4</code> , <code>10*5</code> , <code>10*7</code> , <code>10*8</code>	<code>error_unit_profile_unsupported</code> (§3.5)
Year and month variants	<code>a_t</code> (tropical), <code>a_g</code> (gregorian), <code>mo_s</code> , <code>mo_g</code>	<code>error_unit_profile_unsupported</code> — Bovnar has only the Julian forms

Class	Examples	Outcome
Osmolality	<code>osm</code> , <code>mosm/kg</code>	<code>error_unit_profile_unsupported</code>
Constants as units	<code>[c]</code> , <code>[e]</code> , <code>[k]</code> , <code>[h]</code>	<code>error_unit_profile_unsupported</code>
Over 8 components	any expression exceeding <code>BVNR_MAX_UNIT_COMPONENTS</code>	<code>error_unit_profile_unsupported</code>

The middle rows are the honest cost of the design. A clinical corpus will hit `error_unit_profile_unsupported` on real codes, and the profile does not pretend otherwise: it names the failure rather than accepting the string and leaving the consumer to discover the problem. Which of these should become real native units — the tropical year, osmolality — is a registry question this document does not answer.

7. Data model

7.1 The arbitrary-unit block

UCUM's arbitrary atoms get a contiguous run of `value_base_unit_t` ids appended after the current last enumerator. `src/utis/bvn_internal_dims.h` already documents the id layout and pins `BVN_VALUE_BASE_UNIT_COUNT` to the last enumerator with a compile-time check, so the addition is the ordinary "append and move the check" operation the header describes — the tables sized by that count are indexed by the enum value, and an undersized count is an out-of-bounds read rather than a cosmetic mismatch.

Two constants bracket the run so §7.2 can test membership with two comparisons:

```
#define BVN_UCUM_ARBITRARY_FIRST <first id>
#define BVN_UCUM_ARBITRARY_LAST  <last id>
```

Each entry carries the empty dimension vector, factor `1.0`, `.affine = false`, UCUM's own metric flag as its prefix policy, and **no native alias** — the alias table is what makes a unit reachable from native notation, and leaving it empty is what keeps these profile-only.

7.2 Incommensurability, via the mechanism currencies already use

The design sketched a new predicate in `bvni_dims_match` / `bvni_kinds_match` comparing arbitrary multisets. It was not needed. The library already has a class of units with no SI conversion row — currencies — and the machinery that handles them gives exactly the right answers here.

Two refusals, one each in the two functions everything else is built on:

- `bvn_unit_to_si_factor` sets `*ok = false` on an arbitrary component;

- `bvn_unit_dimension_vector` returns false on one.

`bvn_units_compatible` is built on both, so it reports false for anything containing an arbitrary unit. `bvn_unit_convert_factor` then falls through to `bvn_unit_prefix_only_delta` — the same-unit-apart-from-prefixes path that exists for currencies — and that is what produces the rest:

Pair	Result	Why
<code>ucum:[IU] → ucum:[IU]</code>	factor 1	prefix-only delta of zero
<code>ucum:[IU] → ucum:[PFU]</code>	refused	different bases
<code>ucum:[IU] → mol, %, no_unit</code>	refused	different bases, and no dimension to fall back on
<code>ucum:[IU]/L → ucum:[IU]/mL</code>	factor 1/1000	same bases, prefix delta 3

The last row is the one that matters: `[IU]/L` is a genuine concentration and rescaling its volume is exact. A flat "anything arbitrary is incomparable" rule would have refused it.

Note the shape of the answer: `bvn_units_compatible` says false for `[IU]` against itself, while `bvn_unit_convert_factor` returns 1. That reads oddly until you see it is exactly how `$USD` against `$USD` has always behaved — compatibility is a statement about dimension, and neither a currency nor an assay unit has one.

`bvni_is_arbitrary` is still a two-comparison range test. The block is contiguous and sits above every native unit, and both facts are static assertions in `bvn_internal_dims.h` rather than comments: a native unit appended past the block's first id would otherwise become silently incommensurable with everything.

7.3 No new field on the data event

The design added `unit_source / unit_source_length` to `bvnr_data_t`, following the `frac_data / frac_length` precedent from spec 1.1, so a consumer could see the producer's exact spelling. It did not ship, and `bvnr_data_t` is unchanged.

The reason is the writer, not the reader. Capturing the source text on the way in is easy — the lexer already holds it. Getting it back out is not: every `bvnr_write_*` entry point takes a `value_unit_t`, and a `value_unit_t` has nowhere to put an annotation, so preserving one end to end means a new parameter on the whole writer surface for a string the unit model ignores. That is a large change for §5.2's single loss. It is recorded in §10.4 rather than half-built.

7.4 New error codes

Two, appended after `error_octet_stream_truncated` (48), moving `BVN_ERROR_COUNT` with them — the fuzz harnesses use that count as their bound for "is this a real error code" and trap above it, so a stale count turns a legitimate new error into a fuzz crash.

Code	Meaning
<code>error_unit_profile_unknown</code>	The namespace before the <code>:</code> is not a profile this build supports
<code>error_unit_profile_unsupported</code>	Valid UCUM over known atoms, but no Bovnar representation (§6.4)

Malformed UCUM, and UCUM over an atom UCUM does not define, stay `error_unit_illegal`. The split is what a producer needs: "you wrote it wrong" and "you wrote it right and we cannot carry it" call for different fixes.

8. API

8.1 C

```

/* Unchanged. Dispatches on a "name:" prefix; a native string behaves exactly as
 * it does today. This is the single door – every path that reaches a unit goes
 * through it, so a profile unit cannot arrive by a route that skips a check. */
value_unit_t bvn_parse_unit (const uint8_t* unit, bool* ok);
value_unit_t bvn_parse_unit_n(const uint8_t* unit, uint32_t len, bool* ok);

/* Unchanged. Emits native form, or profile form when a component is
 * profile-only (section 5.1). */
int32_t bvn_unit_to_string(value_unit_t u, char* buf, size_t bufsize);

/* New. Why a rejected unit string is not a unit: error_unit_illegal,
 * error_unit_profile_unknown or error_unit_profile_unsupported. Re-parses, so it
 * is for the error path; a string that DOES parse reports error_none. */
error_code_t bvn_unit_error_code(const uint8_t* unit, uint32_t len);

/* New. True when a unit has no native spelling – it carries a UCUM arbitrary
 * atom – so bvn_unit_to_string emits profile notation. */
bool bvn_unit_is_profile_only(value_unit_t u);

/* New. A UCUM code (without the "ucum:" prefix); negative on failure
 * (section 5.3). */
int32_t bvn_unit_to_ucum(value_unit_t u, char* buf, size_t bufsize);

```

`bvn_parse_unit` keeps its signature, which is why the policy strings of §4.3 work with no change at all: `bvnr_unit_policy_t` parses its targets with it, so the notation arrives for free.

8.2 Python

`parse_unit`, `unit_to_str`, `units_compatible` and `UnitPolicy` accept and produce the profile notation with no signature change. Three additions mirror the C ones:

```

bovnar.unit_to_ucum(vu)      # -> str; raises for a unit with no UCUM code
bovnar.unit_is_profile_only(vu) # -> bool
bovnar.unit_error_code(s)    # -> int (error_code_t), 0 when s parses

```

The existing `from_pint_unit` / `to_pint_unit` bridge is untouched.

8.3 CLI

No new flags. `--unit`, `--field`, `--require-dimension` and `--require-field` take unit strings and therefore take profile strings:

```
bovnar validate --require-field '.patient.systolic=ucum:mm[Hg]' chart.bvnr
bovnar events --unit 'ucum:mmol/L' labs.bvnr
```

`bovnar pretty-print` emits the canonical form (§5.1), which for a translated unit is the native spelling.

9. Build and conformance

9.1 Where the table lives

`src/gendata/ucum.bvnr`, hand-edited, in the same shape as `units.bvnr`, `currencies.bvnr` and `prefixes.bvnr`. Four lists: UCUM's prefix spellings with their decades, the atoms that translate (with their native target expression and UCUM's metric flag), the arbitrary units (with their appended `value_base_unit_t` id), and the atoms that are known but refused (with the reason). `gen_ucum.py` generates the C lookup tables on every build, wired into CMake next to the other three generators; the generated files are never edited.

The native target is stored as **source text**, not as a pre-baked `value_unit_t`, and the C side parses it with the same `bvn_parse_unit` every document goes through. That keeps the generator from needing a second implementation of the unit grammar in Python, which is how a generated table starts disagreeing with the parser it feeds.

9.2 What the generator checks, and what it does not

At build time `gen_ucum.py` refuses to emit a table when:

- an atom appears in more than one of the three lists (an atom has exactly one outcome);
- a prefix declares a decade with no SI prefix — the fold could never discharge it;
- the arbitrary ids are not a contiguous run starting one past the last native unit (the tables sized by `BVN_VALUE_BASE_UNIT_COUNT` are indexed by the enum value, so a gap is a hole of zeroed rows that reads as a valid unit);
- a `.bovnar` target names a prefix or a unit this build's registry does not have.

That last check is the useful one day to day: a typo in a target would otherwise surface as `error_unit_illegal` on a UCUM code that is perfectly valid, which is the most confusing failure this table can produce.

What it does not check is the UCUM side. The design called for the generator to carry UCUM's own value for each atom and refuse any row whose factor and dimension vector did not match the native target exactly — so that no belief about what a UCUM atom is worth stayed load-bearing. That did not ship: `ucum.bvnr` carries the target but not UCUM's value, so the mappings rest on the table author. Where a value was uncertain, or where it was certainly close but not equal to a native unit, the atom went into `.unsupported` instead — which is why `osm`, `[Btu]`, `cal_IT` and `[dr_ap]` are refused rather than mapped onto the nearly-right native unit. §10.2 treats this as the standing risk, and §10.4 records it as unbuilt.

The generator also emits the **reverse** table §5.3 uses, choosing the canonical atom for each base (shortest code, ties alphabetically) and recording that atom's own decade. Deriving it rather than searching the forward table at run time is what makes `bvn_unit_to_ucum` deterministic; the 592 round trips quoted in §5.3 are the check that it agrees with the forward direction.

9.3 Tests

`tests/bovnar_ucum_test.c` (CTest target `bvnr_ucum_test`, labels `unit;si;ucum`) pins the behavioural claims of this document: the three outcomes with their exact error codes, equivalence with the native spelling, UCUM's non-latching `/`, annotation inertness, every worked fold case in §3.5 including the four refused decades, each collision in §6.2, arbitrary-unit incommensurability including the `[IU]/L` ↔ `[IU]/mL` case, profile-only round-trip, and the partiality of `bvn_unit_to_ucum`.

`tests/bovnar_unit_ext_test.c` pins the block boundary: the last native unit sits below `BVN_UCUM_ARBITRARY_FIRST`, and `BVN_VALUE_BASE_UNIT_COUNT` tracks the last arbitrary id.

A conformance-corpus group did not ship. The design called for a `unit-profile-ucum` group in `bvnr_conformance` so a third-party implementation could be held to the same rules; that is a corpus addition, not a library change, and it is listed in §10.4. In particular the error-code move of §2.3 — `<float:64,m[s]>` was `error_unexpected_input_byte` and is now `error_unit_illegal` — is pinned by the unit test but not by the conformance corpus.

9.4 No build switch

The profile is unconditional. There is no `BVNR_WITH_UCUM_PROFILE` option and no addition to a feature-report function: a build that has the profile has all of it. `error_unit_profile_unknown` therefore means only what it says — the namespace is not one this build defines — and today that is every namespace except `ucum`. If the profile ever becomes optional, that error code is already the right answer for a build without it, which is why it exists as a separate code rather than as `error_unit_illegal`.

10. Cost, risk, and what is left out

10.1 What it cost

Area	Change
Lexer	15 state-table entries across three states (§2.3); brace/bracket-depth tracking in the type-parameter scanner (§2.4)
Unit parser	<code>src/utils/bovnar_ucum.c</code> — namespace dispatch, the UCUM expression parser, the translator, the fold
Registry	<code>src/gendata/ucum.bvnr</code> (142 mapped, 32 arbitrary, 51 refused), <code>gen_ucum.py</code> emitting six tables; one appended <code>value_base_unit_t</code> run (397–428); <code>BVN_VALUE_BASE_UNIT_COUNT</code> 397 → 429
Compatibility	Two refusals, one each in <code>bvn_unit_to_si_factor</code> and <code>bvn_unit_dimension_vector</code> (§7.2)
Serialisation	One guard at the head of each of the two formatters (§5.1)
ABI	Two error codes; three new functions. No struct changed
Bindings	Three <code>ctypes</code> declarations and three wrappers
Tests	<code>tests/bovnar_ucum_test.c</code> , 105 assertions; two assertions widened in <code>bovn-ar_unit_ext_test.c</code>

The unit parser is the bulk of it. Everything else is small, and — this is what §1.1 buys — the DOM, the writer, the streaming reader, the policy engine and the CLI needed no work at all, because a translated unit is an ordinary unit. The two formatter guards and the two compatibility refusals are the entire footprint outside the new file.

10.2 What can go wrong

The table rests on its author. §9.2's factor proof did not ship, so every mapping in §6.1 is a claim this repository cannot check. The native side of each is verified — the target parses and its SI factor is asserted in the test — but nothing confirms that UCUM's `[Btu_IT]` really is 1055.05585262 J. The mitigation in place is conservative rather than mechanical: an atom whose value was uncertain went into `.unsupported`, so the failure mode is a refused code rather than a wrong number. That is the right default and it is not a substitute for the check.

The table rots. UCUM revises; `ucum.bvnr` does not, and nothing in the build notices.

The refusal set is where adopters leave. §6.4 refuses osmolality, the non-Julian years, the referenced bels and four decades of scale. A clinical corpus will meet several of those early, and each is a reason to conclude the profile does not really support UCUM. The

counter-argument — that naming a refusal beats accepting a string you cannot reason about — is correct and will not always be persuasive.

Annotation equality will surprise someone. §3.4 is UCUM's rule faithfully applied, and it still means two units a clinician reads as different compare as the same.

One error code moved. `<float:64,m[s]>` was `error_unexpected_input_byte` and is now `error_unit_illegal` (§2.3). Both are refusals of the same document, but a consumer switching on the exact code sees a change.

10.3 Deliberately not attempted

- **Temperature difference.** `Cel` is a scale, here as in UCUM, and a difference of 25 °C still converts as 298.15 K. CF 1.12 closed this with a `units_metadata` attribute carrying `temperature: difference`; Bovnar has no equivalent and this profile does not add one, because a delta scale is a native registry change and importing it through a foreign notation would put the fix somewhere no native document could reach it. It remains the format's most concrete gap, and it is worth more than this whole profile.
- **Case-insensitive UCUM.** UCUM defines a case-insensitive variant. `ucum:` is the case-sensitive one only. `ucum_ci:` is reserved and undefined; it would need its own atom table and would make §6.2's collisions materially worse.
- **CF and UDUNITS.** `cf:` and `udunits:` are reserved by the grammar of §2.6 and specified nowhere. CF's `units` strings are UDUNITS syntax with a separate standard-name table and a reference date embedded in the time unit; none of that fits a per-value unit slot, and the honest answer for CF data is a converter, not a profile.
- **Exchange rates.** Unchanged and unchangeable: currencies carry no conversion table, and a cross-currency conversion is refused rather than guessed (doc/2 §9.6). UCUM has no currencies, so the profile never reaches this.

10.4 Specified here but not built

Four things this document describes are not in the library. They are listed together so the gap is one paragraph to read rather than four sections to cross-check.

Not built	Where it is described	Consequence
Verbatim source preservation (<code>bvnr_data_t.unit_source</code> , writer re-emission)	§5.2, §7.3	An annotation is dropped by a document built through the writer API. A parse-and-re-serialise round trip keeps it
The generator's factor proof against UCUM's own values	§9.2	Every mapping rests on the table author; §10.2
A <code>unit-profile-ucum</code> conformance-corpus group	§9.3	A third-party implementation has no corpus to be held to. The reference implementation is covered by its own test

Not built	Where it is described	Consequence
<code>BVNR_WITH_UCUM_PROFILE</code> and feature reporting	§9.4	The profile is unconditional, which is a simplification rather than a loss

The first two are the ones worth building next, in that order.

See also

- [Unit & Currency Reference](#) — the native registry and notation grammar this profile sits beside
 - [Unit Ambiguities](#) — how a unit token is resolved natively, and the pairs that look interchangeable
 - [Read/Write API](#) — the unit policy a translated unit passes through unchanged
 - [Read/Write API](#) — the data event `unit_source` is added to, and the `want_unit` hook
 - [Conformance Test Tool](#) — where the `unit-profile-ucum` group lives
 - [Unit Cheatsheet](#) — the native spellings the transliteration table targets
-

End of Bovnar — UCUM Unit Profile (Bovnar spec 1.1).