

# Bovnar Quantity Annotation System — Unit and Currency Reference

Bovnar (BVNR) v1.0 documentation · Unit & Currency Reference · 2026-06-01

**Applies to:** Bovnar (BVNR) specification version 1.0 **Scope:** Physical units, currency codes, prefix rules, disambiguation, C/Python APIs, and validation.

## Table of Contents

1. Overview
2. Syntax — Annotation as a Type Parameter
3. Physical Base Units - 3.1 SI Base Units - 3.2 Named SI-Derived Units - 3.3 Non-SI Units Accepted for Use with SI - 3.4 Imperial and US Customary Units - 3.5 Pressure Units - 3.6 Energy Units - 3.7 Power Units - 3.8 Force Units - 3.9 Speed and Rotational Frequency - 3.10 Volume Units - 3.11 Area Units - 3.12 Angle Units - 3.13 CGS Units - 3.14 Radiation Units - 3.15 Logarithmic Units - 3.16 Electrical Power Units - 3.17 Digital Units - 3.18 Textile Linear Density - 3.19 US Apothecary / Dry Volume - 3.20 Old German Units - 3.21 Additional Length Units - 3.22 Additional Mass Units - 3.23 Acceleration - 3.24 Signal Rate - 3.25 Ratio and Proportion Units - 3.26 Sentinel Value
4. Prefixes - 4.1 SI Prefixes - 4.2 IEC Binary Prefixes
5. Unit Notation Grammar - 5.1 Simple Units - 5.2 Compound Units - 5.3 Separators - 5.4 Denominator Semantics
6. Exponents - 6.1 Unicode Superscript Form - 6.2 ASCII Caret Form - 6.3 Exponent Edge Cases
7. The `no_unit` Keyword
8. Constraints and Limits
9. Currency Codes - 9.1 The `$` Sigil Rule - 9.2 ISO 4217 Fiat Currencies and Precious Metals - 9.3 Cryptocurrencies - 9.4 Prefix Rules for Currency Units - 9.5 Compound Currency Expressions - 9.6 Compatibility Rules - 9.7 Type Pairing Recommendations
10. Symbol Disambiguation
  - 10.1 The Namespace Rule as Disambiguator
  - 10.2 Exhaustive Conflict Table
  - 10.3 The CUP Case in Detail
  - 10.4 The Mandatory Currency Sigil
11. C Data Model
  - 11.1 Enumerations
  - 11.2 Structures
  - 11.3 Convenience Macros

- 12. C API Functions
  - 12.1 Parsing a Unit String
  - 12.2 Serializing a Unit
  - 12.3 Prefix Factor and Exponent Queries
  - 12.4 SI Conversion API
  - 12.5 Currency API
  - 12.6 Python API
- 13. Integration with the Parser Event Stream
- 14. Validation Errors
- 15. Annotated Examples
  - 15.1 Physical Quantities
  - 15.2 Digital Storage
  - 15.3 Compound SI Quantities
  - 15.4 Currency Amounts and Rates
  - 15.5 Error Cases

## 1. Overview

The Bovnar quantity annotation system is an **optional, per-value annotation** that attaches a physical unit or currency denomination to any numeric field. It is part of the type annotation ( `<family:width,_base,unit>` ) and applies to the `uint`, `sint`, `float`, `float_fix`, and `float_dec` type families.

Two distinct namespaces share the annotation slot:

- **Physical units** — 163 named base units covering SI, Imperial, CGS, radiation, surveying, culinary, Old German, and digital storage quantities.
- **Currency codes** — 214 monetary denominations: 164 ISO 4217 alphabetic codes (including precious-metal X-codes; 2 are historical: HRK retired 2023-01-01, SLL replaced by SLE 2022) and 50 cryptocurrency tickers.

Both namespaces are syntactically unified: the same grammar, the same `~` prefix separator, the same compound-unit operators ( `.`, `*`, `/` ), and the same `value_unit_t` data model apply to both. They are separated purely by a token-classification rule described in §9.1 and §10.

Annotations are **descriptive**, not prescriptive. Bovnar validates form and type; it does not perform dimensional analysis, unit conversion, or exchange-rate arithmetic. That responsibility belongs to the consuming application.

## Design Principles

- **SI-first.** All SI base units, all 22 BIPM-2019 named derived units, and all 24 current SI prefixes (quecto ... quetta) are supported.

- **Binary-prefix aware.** IEC 80000-13 binary prefixes (kibi ... quebi) are supported for digital storage quantities.
- **Compound units.** Derived quantities (m/s, kg·m/s<sup>2</sup>, USD/oz\_t) are expressed inline without separate schema definitions.
- **Two exponent notations.** Unicode superscript ( $\text{m}^2$ ,  $\text{s}^{-2}$ ) and ASCII caret ( $\text{m}^2$ ,  $\text{s}^{-2}$ ) are accepted equivalently.
- **Currency as a first-class unit.** ISO 4217 and cryptocurrency codes participate in all unit composition rules — prefixes, compound expressions, and the `value_unit_t` representation — with no special-case parsing.
- **Dimensionless values.** The keyword `no_unit` is the canonical representation of a dimensionless quantity.

## 2. Syntax — Annotation as a Type Parameter

The unit or currency code occupies the **third positional parameter class** of a type annotation, after the optional bit-width and optional base. Parameter classes are identified by their content, not by position:

```
type-spec      = param-type [ ":" type-param-list ]
type-param-list = type-param { "," type-param }
type-param     = width-param  (* plain decimal integer, e.g. 32    *)
                | base-param   (* "_" + decimal integer,   e.g. _16  *)
                | unit-param   (* everything else,      e.g. m/s  *)
```

### 2.1 Parameter Ordering Flexibility

```
.val = <uint:32,_10,no_unit> 42;
#      <uint:_10,no_unit,32>      - identical (parameter order is free)
#      <uint:no_unit,_10,32>      - identical
```

### 2.2 Inline Unit Suffix

A unit may be written directly after a scalar value literal, between the value and the terminating `;`:

```
.distance = 1500 m;           # inline physical unit
.speed    = 9.81 m/s;         # compound inline unit
.price     = 19.99 $USD;       # inline currency (mandatory $ sigil)
.gold_rate = 2351.40 $USD/oz_t; # inline compound currency/unit
.ratio     = 3.14 no_unit;     # explicit dimensionless
```

The inline unit uses the **same character set** and **same semantic parser** (`bnv_parse_unit`) as the type-annotation unit parameter. It is terminated by ASCII whitespace, `#` (comment), or `;`.

## Constraints

Situation	Result
No annotation unit; inline unit present	Inline unit becomes the effective unit
Annotation has no unit; inline unit present	Inline unit becomes the effective unit
Annotation unit present; no inline unit	Annotation unit is the effective unit
Annotation unit <b>equals</b> inline unit	Valid; the common unit is used
Annotation unit <b>differs</b> from inline unit	<code>error_unit_mismatch</code>
Inline unit inside an array element	<code>error_unexpected_input_byte</code>

When both are present, equality is checked after parsing via `memcmp` on the complete `value_unit_t` structure. Two strings match if and only if `bvn_parse_unit` produces bit-for-bit identical `value_unit_t` values — so logically equivalent notations (e.g. `m·s-1` vs `m/s`) compare as equal.

```
.v = <float:64,m/s> 9.81 m·s-1;    # OK: both parse to m/s
.v = <float:64,m> 1.0 s;           # ERROR: error_unit_mismatch
```

## Applicable Type Families

Type family	Unit / currency parameter
<code>uint</code>	Supported
<code>sint</code>	Supported
<code>float</code>	Supported (binary floating-point; discouraged for monetary amounts — see §9.7)
<code>float_fix</code>	Supported (wrong for monetary values — see §9.7)
<code>float_dec</code>	Supported; <b>recommended</b> for monetary amounts
<code>utf8</code>	Parameterless: a unit (or any other parameter) is <code>error_illegal_value_type</code>

## 3. Physical Base Units

Bovnar supports 163 named physical base units. Currency codes are a separate namespace and are covered in §9.

**Reading this section:** The *Symbol* column gives the canonical serialized form. *Long forms* are accepted on input but never produced on output. *Enum value* is the `value_base_unit_t` constant used in the C API.

### 3.1 SI Base Units

Symbol	Long forms	Name	Enum value	Notes
s	sec , second , seconds	second	bu_second	SI base unit of time
m	meter , metre , meters , metres	meter	bu_meter	SI base unit of length
g	gram , grams	gram	bu_gram	SI base unit of mass is kg; g carries the prefix
A	amp , amps , ampere , amperes	ampere	bu_ampere	SI base unit of electric current
K	kelvin , kelvins	kelvin	bu_kelvin	SI base unit of thermodynamic temperature
mol	mole , moles	mole	bu_mol	SI base unit of amount of substance
cd	candela , candelas	candela	bu_candela	SI base unit of luminous intensity

**Note on the kilogram:** Bovnar uses g (gram) as the base unit symbol so that the k~ (kilo) prefix can be attached explicitly: k~g = kilogram. This is consistent with how the SI formally defines the kilogram as a prefixed gram.

### 3.2 Named SI-Derived Units

Symbol	Long forms	Name	Enum value	SI Definition
Hz	hertz	hertz	bu_hertz	$s^{-1}$
N	newton , newtons	newton	bu_newton	$kg \cdot m \cdot s^{-2}$
Pa	pascal , pascals	pascal	bu_pascal	$kg \cdot m^{-1} \cdot s^{-2}$
J	joule , joules	joule	bu_joule	$kg \cdot m^2 \cdot s^{-2}$
W	watt , watts	watt	bu_watt	$kg \cdot m^2 \cdot s^{-3}$
V	volt , volts	volt	bu_volt	$kg \cdot m^2 \cdot A^{-1} \cdot s^{-3}$
$\Omega$	ohm , ohms , Ohm	ohm	bu_ohm	$kg \cdot m^2 \cdot A^{-2} \cdot s^{-3}$ — U+2126 OHM SIGN, UTF-8: 0xE2 0x84 0xA6 ; U+03A9 (Greek capital omega) <b>not</b> accepted
F	farad , farads	farad	bu_farad	$kg^{-1} \cdot m^{-2} \cdot A^2 \cdot s^4$
C	coulomb , coulombs	coulomb	bu_coulomb	$A \cdot s$
S	siemens	siemens	bu_siemens	$kg^{-1} \cdot m^{-2} \cdot A^2 \cdot s^3$
Wb	weber , webers	weber	bu_weber	$kg \cdot m^2 \cdot A^{-1} \cdot s^{-2}$
T	tesla , teslas	tesla	bu_tesla	$kg \cdot A^{-1} \cdot s^{-2}$
H	henry , henrys , henries	henry	bu_henry	$kg \cdot m^2 \cdot A^{-2} \cdot s^{-2}$

Symbol	Long forms	Name	Enum value	SI Definition
°C	degC , degrC , degreeC , degreesC , celsius	degree Celsius	bu_celsius	K = °C + 273.15 (affine); BIPM Table 4 entry 14
lm	lumen , lumens	lumen	bu_lumen	cd·sr
lx	lux	lux	bu_lux	cd·sr·m <sup>-2</sup>
Bq	becquerel , becquerels	becquerel	bu_becquerel	s <sup>-1</sup>
Gy	gray , grays	gray	bu_gray	m <sup>2</sup> ·s <sup>-2</sup>
Sv	sievert , sieverts	sievert	bu_sievert	m <sup>2</sup> ·s <sup>-2</sup>
kat	katal , katal	katal	bu_katal	mol·s <sup>-1</sup>
rad	radian , radians	radian	bu_radian	dimensionless (plane angle; m/m)
sr	steradian , steradians	steradian	bu_steradian	dimensionless (solid angle; m <sup>2</sup> /m <sup>2</sup> )

### 3.3 Non-SI Units Accepted for Use with SI

Symbol	Long forms	Name	Enum value	Notes
L , l	liter , litre , liters , litres	liter	bu_liter	10 <sup>-3</sup> m <sup>3</sup>
min	minute , minutes	minute	bu_minute	60 s
h	hour , hours	hour	bu_hour	3600 s
d	day , days	day	bu_day	86400 s
wk	week , weeks	week	bu_week	604800 s
yr	year , years	year	bu_year	31557600 s (Julian year)
mo	month , months	month (Julian)	bu_month	2629800 s (= 365.25 d / 12)
fn	fortnight , fortnights	fortnight	bu_fortnight	1209600 s (= 14 d)
° , deg	degr , degree , degrees	degree (angle)	bu_degree	π/180 rad — U+00B0
t	tonne	tonne	bu_tonne	10 <sup>3</sup> kg
bar	—	bar	bu_bar	10 <sup>5</sup> Pa
eV	electronvolt	electronvolt	bu_electronvolt	1.602176634×10 <sup>-19</sup> J
Da	dalton	dalton	bu_dalton	1.66053906660×10 <sup>-27</sup> kg
au	—	astronomical unit	bu_astronomical_unit	1.495978707×10 <sup>11</sup> m
ha	hectare	hectare	bu_hectare	10 <sup>4</sup> m <sup>2</sup>

## 3.4 Imperial and US Customary Units

### Length

Symbol	Long forms	Name	Enum value	Factor
<code>in</code>	<code>inch</code> , <code>inches</code>	inch	<code>bu_inch</code>	0.0254 m (exact)
<code>ft</code>	<code>foot</code> , <code>feet</code>	foot	<code>bu_foot</code>	0.3048 m (exact)
<code>yd</code>	<code>yard</code> , <code>yards</code>	yard	<code>bu_yard</code>	0.9144 m (exact)
<code>mi</code>	<code>mile</code> , <code>miles</code>	statute mile	<code>bu_mile</code>	1609.344 m (exact)
<code>nmi</code>	<code>nautical_mile</code> , <code>nautical_miles</code>	nautical mile	<code>bu_nautical_mile</code>	1852 m (exact)
<code>Å</code> (U+212B)	<code>angstrom</code> , <code>angstroms</code> , <code>Å</code> (U+00C5)	ångström	<code>bu_angstrom</code>	10 <sup>-10</sup> m
<code>ly</code>	<code>light_year</code> , <code>light_years</code>	light-year	<code>bu_light_year</code>	9.4607304725808×10 <sup>15</sup> m
<code>pc</code>	<code>parsec</code> , <code>parsecs</code>	parsec	<code>bu_parsec</code>	3.085677581491367×10 <sup>16</sup> m
<code>fur</code>	<code>furlong</code> , <code>furlongs</code>	furlong	<code>bu_furlong</code>	201.168 m (exact)
<code>fath</code>	<code>fathom</code> , <code>fathoms</code>	fathom	<code>bu_fathom</code>	1.8288 m (exact)
<code>thou</code>	<code>thou</code> , <code>mil</code> , <code>mils</code>	thou	<code>bu_thou</code>	25.4×10 <sup>-6</sup> m (exact)
<code>ch</code>	<code>chain</code> , <code>chains</code>	chain (Gunter's)	<code>bu_chain</code>	20.1168 m (exact)
<code>rd</code>	<code>rod</code> , <code>rods</code>	rod (pole, perch)	<code>bu_rod</code>	5.0292 m (exact)

**Thou vs mil:** Both `thou` and `mil` are accepted for 1/1000 of an inch (25.4 μm). The canonical output form is `thou`. Note that `mil` does **not** mean milliradian; milliradians are written `m~rad`.

### Mass

Symbol	Long forms	Name	Enum value	Factor
<code>lb</code>	<code>lbs</code> , <code>pound</code> , <code>pounds</code>	pound (avoirdupois)	<code>bu_pound</code>	0.45359237 kg (exact)
<code>oz</code>	<code>ounce</code> , <code>ounces</code>	ounce (avoirdupois)	<code>bu_ounce</code>	0.028349523125 kg (exact)
<code>gr</code>	<code>grain</code> , <code>grains</code>	grain	<code>bu_grain</code>	6.479891×10 <sup>-5</sup> kg (exact)
<code>st</code>	<code>stone</code> , <code>stones</code>	stone	<code>bu_stone</code>	6.35029318 kg (exact)
<code>tn_sh</code>	<code>short_ton</code> , <code>short_tons</code>	short ton (US ton)	<code>bu_short_ton</code>	907.18474 kg (exact)

Symbol	Long forms	Name	Enum value	Factor
<code>tn_l</code>	<code>long_ton</code> , <code>long_tons</code>	long ton (UK ton)	<code>bu_long_ton</code>	1016.0469088 kg (exact)
<code>oz_t</code>	<code>troy_ounce</code> , <code>troy_ounces</code>	troy ounce	<code>bu_troy_ounce</code>	0.0311034768 kg (exact)
<code>ct</code>	<code>carat</code> , <code>carats</code>	metric carat	<code>bu_carat</code>	$2 \times 10^{-4}$ kg (exact)
<code>slug</code>	<code>slugs</code>	slug	<code>bu_slug</code>	14.593902937 kg
<code>dr</code>	<code>dram</code> , <code>drams</code>	dram (avoirdupois)	<code>bu_dram</code>	$1.7718451953125 \times 10^{-3}$ kg (exact)
<code>dwt</code>	<code>pennyweight</code> , <code>pennyweights</code>	pennyweight (troy)	<code>bu_pennyweight</code>	$1.55517384 \times 10^{-3}$ kg (exact)

## Temperature

Symbol	Long forms	Name	Enum value	Conversion
<code>°C</code> , <code>degC</code>	<code>degrC</code> , <code>degreeC</code> , <code>degreesC</code> , <code>celsius</code>	degree Celsius	<code>bu_celsius</code>	$K = ^\circ C + 273.15$ (affine) — also §3.2 (BIPM named derived unit)
<code>°F</code> , <code>degF</code>	<code>degrF</code> , <code>degreeF</code> , <code>degreesF</code> , <code>fahrenheit</code>	degree Fahrenheit	<code>bu_fahrenheit</code>	$K = (^\circ F + 459.67) \times 5/9$ (affine)
<code>°Ra</code> , <code>degRa</code>	<code>degrRa</code> , <code>degreeRa</code> , <code>degreesRa</code> , <code>rankine</code>	degree Rankine	<code>bu_rankine</code>	$K = ^\circ Ra \times 5/9$ (linear)
<code>°De</code> , <code>degDe</code>	<code>degrDe</code> , <code>degreeDe</code> , <code>degreesDe</code> , <code>delisle</code>	degree Delisle	<code>bu_delisle</code>	$K = 373.15 - ^\circ De \times 2/3$ (affine)
<code>°N</code> , <code>degN</code>	<code>degrN</code> , <code>degreeN</code> , <code>degreesN</code> , <code>newton_temperature</code>	degree Newton	<code>bu_newton_temp</code>	$K = ^\circ N \times 100/33 + 273.15$ (affine)
<code>°Re</code> , <code>degRe</code>	<code>degrRe</code> , <code>degreeRe</code> , <code>degreesRe</code> , <code>reaumur</code>	degree Réaumur	<code>bu_reaumur</code>	$K = ^\circ Re \times 5/4 + 273.15$ (affine)
<code>°Ro</code> , <code>degRo</code>	<code>degrRo</code> , <code>degreeRo</code> , <code>degreesRo</code> , <code>romer</code>	degree Rømer	<code>bu_romer</code>	$K = (^\circ Ro - 7.5) \times 40/21 + 273.15$ (affine)

Kelvin (`K`) is the SI base unit (§3.1). `Ra` not `R` — `R` is reserved for the röntgen (`bu_roentgen`).

## 3.5 Pressure Units

Symbol	Long forms	Name	Enum value	Factor
<code>atm</code>	<code>atmosphere</code> , <code>atmospheres</code>	standard atmosphere	<code>bu_atmosphere</code>	101325 Pa (exact)
<code>at</code>	<code>atmosphere_technical</code>	atmosphere technical	<code>bu_atmosphere_technical</code>	98066.5 Pa (= 1 kgf/cm <sup>2</sup> )



Symbol	Long forms	Name	Enum value	Factor
mmHg	—	millimetre of mercury	bu_mmhg	133.322387415 Pa
Torr	torr	torr	bu_torr	101325/760 Pa
psi	—	pound-force per square inch	bu_psi	6894.757293168361 Pa
inHg	inch_hg , inch_mercury	inch of mercury	bu_inch_hg	3386.388645 Pa

### 3.6 Energy Units

Symbol	Long forms	Name	Enum value	Factor
cal	calorie , calories	thermochemical calorie	bu_calorie	4.184 J (exact)
Btu	btu	International Table BTU	bu_btu	1055.05585262 J
erg	ergs	erg	bu_erg	10 <sup>-7</sup> J (exact)
thm	therm , therms	US therm	bu_therm	1.05480400×10 <sup>8</sup> J (exact)
ft_lb	foot_pound , foot_pounds	foot-pound	bu_foot_pound	1.3558179483 J

**BTU alias note:** BTU (all uppercase, three characters) is a valid alias for bu\_btu. Because currencies require the \$ sigil, the bare token BTU is a physical-unit lookup and resolves to bu\_btu; Btu and btu are also accepted. See §10.2 for the complete look-alike table.

### 3.7 Power Units

Symbol	Long forms	Name	Enum value	Factor
hp	horsepower	mechanical horsepower	bu_horsepower	745.69987158227 W
PS	CV , metric_horsepower	metric horsepower	bu_metric_horsepower	735.49875 W (exact)

### 3.8 Force Units

Symbol	Long forms	Name	Enum value	Factor
lbf	pound_force	pound-force	bu_pound_force	4.4482216152605 N
dyn	dyne , dynes	dyne	bu_dyne	10 <sup>-5</sup> N (exact)
kip	kips	kip (kilopound-force)	bu_kip	4448.2216152605 N

Symbol	Long forms	Name	Enum value	Factor
<code>kgf</code>	<code>kilogram_force</code>	kilogram-force	<code>bu_kilogram_force</code>	9.80665 N (exact)

### 3.9 Speed and Rotational Frequency Units

Symbol	Long forms	Name	Enum value	Factor
<code>kn</code>	<code>knot</code> , <code>knots</code>	knot	<code>bu_knot</code>	1852/3600 m/s
<code>rpm</code>	—	revolutions per minute	<code>bu_rpm</code>	1/60 s <sup>-1</sup>

`kn` has dimension m·s<sup>-1</sup>. `rpm` has dimension s<sup>-1</sup> (rotational frequency, not linear speed); it is grouped here by convention.

### 3.10 Volume Units

#### US Liquid Volume

Symbol	Long forms	Name	Enum value	Factor
<code>gal</code>	<code>gallon</code> , <code>gallons</code>	US liquid gallon	<code>bu_gallon</code>	3.785411784×10 <sup>-3</sup> m <sup>3</sup> (exact)
<code>qt</code>	<code>quart</code> , <code>quarts</code>	US liquid quart	<code>bu_quart</code>	9.46352946×10 <sup>-4</sup> m <sup>3</sup>
<code>pt</code>	<code>pint</code> , <code>pints</code>	US liquid pint	<code>bu_pint</code>	4.73176473×10 <sup>-4</sup> m <sup>3</sup>
<code>cup</code>	<code>cups</code>	US cup	<code>bu_cup</code>	2.365882365×10 <sup>-4</sup> m <sup>3</sup>
<code>gi</code>	<code>gill</code> , <code>gills</code>	US gill	<code>bu_gill</code>	1.18294118250×10 <sup>-4</sup> m <sup>3</sup>
<code>fl_oz</code>	<code>fluid_ounce</code> , <code>fluid_ounces</code>	US fluid ounce	<code>bu_fluid_ounce</code>	2.95735295625×10 <sup>-5</sup> m <sup>3</sup>
<code>tbsp</code>	<code>tablespoon</code> , <code>tablespoons</code>	US tablespoon	<code>bu_tablespoon</code>	1.47867648×10 <sup>-5</sup> m <sup>3</sup>
<code>tsp</code>	<code>teaspoon</code> , <code>teaspoons</code>	US teaspoon	<code>bu_teaspoon</code>	4.92892159375×10 <sup>-6</sup> m <sup>3</sup>
<code>bbl</code>	<code>barrel</code> , <code>barrels</code>	petroleum barrel	<code>bu_barrel</code>	0.158987294928 m <sup>3</sup>

**`cup` disambiguation:** The canonical symbol for the US cup volume unit is `cup` (all lowercase). The Cuban Peso (ISO 4217 code 192) is written with the mandatory currency sigil as `$CUP`; the bare uppercase token `CUP` is `error_unit_illegal`. The two cannot be confused. See §10.3 for full details.

#### UK Imperial Volume

Symbol	Long forms	Name	Enum value	Factor
<code>gal_uk</code>	<code>gallon_uk</code> , <code>gallons_uk</code>	imperial gallon	<code>bu_gallon_uk</code>	4.54609×10 <sup>-3</sup> m <sup>3</sup> (exact)

Symbol	Long forms	Name	Enum value	Factor
qt_uk	quart_uk, quarts_uk	imperial quart	bu_quart_uk	$1136.5225 \times 10^{-6} \text{ m}^3$
pt_uk	pint_uk, pints_uk	imperial pint	bu_pint_uk	$568.26125 \times 10^{-6} \text{ m}^3$
gi_uk	gill_uk, gills_uk	imperial gill	bu_gill_uk	$1.420653125 \times 10^{-4} \text{ m}^3$ (exact)
fl_oz_uk	fluid_ounce_uk, fluid_ounces_uk	imperial fluid ounce	bu_fluid_ounce_uk	$28.4130625 \times 10^{-6} \text{ m}^3$

### 3.11 Area Units

Symbol	Long forms	Name	Enum value	Factor
ac	acre, acres	acre	bu_acre	$4046.8564224 \text{ m}^2$ (exact)
barn	barns	barn	bu_barn	$10^{-28} \text{ m}^2$ (exact)

### 3.12 Angle Units

Symbol	Long forms	Name	Enum value	Factor
arcmin	arcminute, arcminutes	arcminute	bu_arcminute	$\pi/10800 \text{ rad}$
arcsec	arcsecond, arcseconds	arcsecond	bu_arcsecond	$\pi/648000 \text{ rad}$
grad	gradian, gradians, gon	gradian	bu_grad	$\pi/200 \text{ rad}$
rev	turn, revolution, revolutions, turns	revolution	bu_revolution	$2\pi \text{ rad}$

### 3.13 CGS Units

Symbol	Long forms	Name	Enum value	SI equivalent
P	poise, poises	poise (dynamic viscosity)	bu_poise	$0.1 \text{ Pa}\cdot\text{s}$
St	stokes, stoke	stokes (kinematic viscosity)	bu_stokes	$10^{-4} \text{ m}^2\cdot\text{s}^{-1}$
G	gauss	gauss (magnetic flux density)	bu_gauss	$10^{-4} \text{ T}$
Mx	maxwell, maxwells	maxwell (magnetic flux)	bu_maxwell	$10^{-8} \text{ Wb}$
Oe	oersted, oersteds	oersted (magnetic field strength)	bu_oersted	$1000/(4\pi) \text{ A/m}$
sb	stilb, stilbs	stilb (luminance)	bu_stilb	$10^4 \text{ cd/m}^2$
ph	phot, photos	phot (illuminance)	bu_phot	$10^4 \text{ lx}$
Gal	galileo, galileos	galileo (acceleration)	bu_galileo	$10^{-2} \text{ m/s}^2$

### 3.14 Radiation Units

Symbol	Long forms	Name	Enum value	SI equivalent
Ci	curie, curies	curie (radioactivity)	bu_curie	$3.7 \times 10^{10} \text{ Bq}$
R	roentgen, roentgens	röntgen (radiation exposure)	bu_roentgen	$2.58 \times 10^{-4} \text{ C/kg}$

Symbol	Long forms	Name	Enum value	SI equivalent
rem	rems	rem (dose equivalent)	bu_rem	10 <sup>-2</sup> Sv

### 3.15 Logarithmic Units

Symbol	Long forms	Name	Enum value	Notes
Np	neper, nepers	neper	bu_neper	dimensionless logarithmic ratio
dB	decibel, decibels	decibel	bu_decibel	1 Np = 20/ln(10) dB ≈ 8.686 dB

### 3.16 Electrical Power Units

Symbol	Long forms	Name	Enum value	Notes
var	vars	var (volt-ampere reactive)	bu_var	reactive power; same SI dimension as W
VA	volt_ampere, volt_amperes	volt-ampere	bu_volt_ampere	apparent power; same SI dimension as W

**Watt, var, and VA:** All three carry the same SI dimensional signature (kg·m<sup>2</sup>·s<sup>-3</sup>). `bvn_units_compatible` returns `true` when comparing them. They are kept as distinct base units because they represent distinct AC power interpretations.

### 3.17 Digital Units

Symbol	Long forms	Name	Enum value
b	bit, bits	bit	bu_bit
B	byte, bytes, Byte, Bytes	byte	bu_byte

### 3.18 Textile Linear Density

Symbol	Long forms	Name	Enum value	Factor
tex	—	tex	bu_tex	1×10 <sup>-6</sup> kg/m (ISO 1144)
den	denier, deniers	denier	bu_denier	1/9000000 kg/m

### 3.19 US Apothecary / Dry Volume

Symbol	Long forms	Name	Enum value	Factor
fl_dr	fluid_dram, fluid_drams	US fluid dram	bu_fluid_dram	3.6966911953125×10 <sup>-6</sup> m <sup>3</sup>
minim	minims	US minim	bu_minim	6.16115199218750×10 <sup>-8</sup> m <sup>3</sup>
pk	peck, pecks	US dry peck	bu_peck	8.80976754172×10 <sup>-3</sup> m <sup>3</sup>
bsh	bushel, bushels	US bushel	bu_bushel	3.523907016688×10 <sup>-2</sup> m <sup>3</sup>

`minim` (not `min`, which is the minute) avoids ambiguity.

### 3.20 Old German Units

Old German units fall into metric-compatible units (still in use in DACH regions) and historical pre-metric Prussian units. The prefix `pr` is reserved for Prussian symbols. No German unit accepts any non-trivial SI or IEC prefix; `bnv_prefix_unit_valid` rejects any non-`si_none` / `iec_none` prefix for every unit from `bu_pfund` through `bu_scheffel`.

#### Metric-Compatible German Units — Mass

Symbol	Long forms	Name	Enum value	Factor
<code>Pfd</code>	<code>pfund</code> , <code>pfunds</code>	Pfund	<code>bu_pfund</code>	0.5 kg (exact)
<code>Ztr</code>	<code>zentner</code>	Zentner	<code>bu_zentner</code>	50 kg (exact)
<code>dz</code>	<code>doppelzentner</code>	Doppelzentner	<code>bu_doppelzentner</code>	100 kg (exact)
<code>lot</code>	<code>lots</code>	Lot	<code>bu_lot</code>	15.625×10 <sup>-3</sup> kg (exact)

#### Historical German Units — Length (Prussian)

Symbol	Long forms	Name	Enum value	Factor
<code>prln</code>	<code>prussian_line</code> , <code>linie</code>	Prussian line	<code>bu_prussian_line</code>	2.17953×10 <sup>-3</sup> m
<code>prz</code>	<code>prussian_zoll</code> , <code>zoll</code>	Prussian Zoll	<code>bu_prussian_zoll</code>	2.61544×10 <sup>-2</sup> m
<code>prf</code>	<code>prussian_fuss</code> , <code>preussischer_fuss</code>	Prussian Fuß	<code>bu_prussian_fuss</code>	3.13853×10 <sup>-1</sup> m
<code>elle</code>	<code>prussian_elle</code> , <code>preussische_elle</code>	Prussian Elle	<code>bu_prussian_elle</code>	6.67160×10 <sup>-1</sup> m
<code>rute</code>	<code>prussian_rute</code> , <code>preussische_rute</code>	Prussian Rute	<code>bu_prussian_rute</code>	3.76624 m
<code>klafter</code>	<code>prussian_klafter</code>	Klafter	<code>bu_klafter</code>	1.88312 m
<code>dt_mi</code>	<code>deutsche_meile</code> , <code>german_mile</code>	Geographische Meile	<code>bu_german_mile</code>	7420.44 m

#### Historical German Units — Area (Prussian)

Symbol	Long forms	Name	Enum value	Factor
<code>morgen</code>	<code>prussian_morgen</code>	Morgen (Prussian)	<code>bu_morgen</code>	2553.22 m <sup>2</sup>

#### Historical German Units — Volume (Prussian)

Symbol	Long forms	Name	Enum value	Factor
<code>schffl</code>	<code>scheffel</code> , <code>prussian_scheffel</code>	Scheffel (Prussian)	<code>bu_scheffel</code>	54.961×10 <sup>-3</sup> m <sup>3</sup>

The enum values for German units occupy positions **348–360**, placed after the entire currency range (134–347). Additional physical units (survey foot, league, cable, hand, quintal, scruple, baud) occupy positions **361–367**. Historical temperature scales (Delisle, Newton, Réaumur, Rømer) occupy positions **368–371**, and the dimensionless ratio units ( `bu_percent` ... `bu_ppb` ) occupy positions **372–377**.

`BVN_VALUE_BASE_UNIT_COUNT` is a `#define` equal to **378** (verified by static assert `bu_ppb + 1 == 378` ). Currencies begin at 134, immediately after the last non-German physical unit.

### 3.21 Additional Length Units

Symbol	Long forms	Name	Enum value	Factor
<code>ftUS</code>	<code>survey_foot</code>	US survey foot	<code>bu_survey_foot</code>	1200/3937 m $\approx$ 0.304800609... m
<code>lea</code>	<code>league</code> , <code>leagues</code>	statute league	<code>bu_league</code>	4828.032 m (= 3 statute miles)
<code>cbl</code>	<code>cable</code> , <code>cables</code>	cable length	<code>bu_cable</code>	185.2 m
<code>hand</code>	<code>hands</code>	hand	<code>bu_hand</code>	0.1016 m (= 4 in, exact)

`ftUS` (US survey foot) differs from `ft` (international foot, 0.3048 m exactly) by about 2 ppm. Used in US geodetic surveying.

### 3.22 Additional Mass Units

Symbol	Long forms	Name	Enum value	Factor
<code>qntl</code>	<code>quintal</code> , <code>quintals</code>	quintal	<code>bu_quintal</code>	100 kg (exact)
<code>sc</code>	<code>scruple</code> , <code>scruples</code>	apothecary scruple	<code>bu_scruple</code>	$1.2959782 \times 10^{-3}$ kg (= 20 grains)

### 3.23 Acceleration

Symbol	Long forms	Name	Enum value	Factor
<code>gn</code>	<code>standard_gravity</code>	standard gravity	<code>bu_standard_gravity</code>	$9.80665 \text{ m} \cdot \text{s}^{-2}$ (exact, CGPM 1901)

### 3.24 Signal Rate

Symbol	Long forms	Name	Enum value	Notes
<code>Bd</code>	<code>baud</code> , <code>bauds</code>	baud	<code>bu_baud</code>	1 symbol/s = $1 \text{ s}^{-1}$ (ITU-T V.662)

### 3.25 Ratio and Proportion Units

Dimensionless scaling factors. A value carrying one of these units reduces to the numeric value multiplied by the factor in the canonical (dimensionless) base representation — e.g. `5 %`  $\equiv$  `0.05`, `250 ppm`  $\equiv$  `0.00025`. Like the other dimensionless ratios they carry an empty dimension vector, but unlike `rad / sr` they do **not** accept SI or IEC prefixes (a prefixed `%` is meaningless).

Symbol	Long forms	Name	Enum value	Factor
<code>%</code>	<code>percent</code>	per cent	<code>bu_percent</code>	$10^{-2}$
<code>‰</code>	<code>per_mille</code>	per mille	<code>bu_per_mille</code>	$10^{-3}$
<code>‱</code>	<code>per_myriad</code>	per myriad	<code>bu_per_myriad</code>	$10^{-4}$
<code>pcm</code>	<code>per_cent_mille</code>	per cent mille	<code>bu_per_cent_mille</code>	$10^{-5}$
<code>ppm</code>	—	parts per million	<code>bu_ppm</code>	$10^{-6}$
<code>ppb</code>	—	parts per billion	<code>bu_ppb</code>	$10^{-9}$

### 3.26 Sentinel Value

`bu_none` (value `0`) is the internal representation of "no base unit", used for the `no_unit` keyword and as the default when no unit annotation is present.

## 4. Prefixes

Prefixes are attached to a base unit symbol with a mandatory `~` separator: `prefix-baseunit`.

### 4.1 SI Prefixes

All 24 current SI prefixes are supported, from quecto ( $10^{-30}$ ) to quetta ( $10^{30}$ ).

Prefix	Symbol	Factor	Enum value
quetta	<code>Q</code>	$10^{30}$	<code>si_quetta</code>
ronna	<code>R</code>	$10^{27}$	<code>si_ronna</code>
yotta	<code>Y</code>	$10^{24}$	<code>si_yotta</code>
zetta	<code>Z</code>	$10^{21}$	<code>si_zetta</code>
exa	<code>E</code>	$10^{18}$	<code>si_exa</code>
peta	<code>P</code>	$10^{15}$	<code>si_peta</code>
tera	<code>T</code>	$10^{12}$	<code>si_tera</code>
giga	<code>G</code>	$10^9$	<code>si_giga</code>
mega	<code>M</code>	$10^6$	<code>si_mega</code>

Prefix	Symbol	Factor	Enum value
kilo	<b>k</b>	$10^3$	<code>si_kilo</code>
hecto	<b>h</b>	$10^2$	<code>si_hecto</code>
deca	<b>da</b>	$10^1$	<code>si_deca</code>
(none)	—	$10^0$	<code>si_none</code>
deci	<b>d</b>	$10^{-1}$	<code>si_deci</code>
centi	<b>c</b>	$10^{-2}$	<code>si_cent</code>
milli	<b>m</b>	$10^{-3}$	<code>si_milli</code>
micro	<b>μ</b>	$10^{-6}$	<code>si_micro</code>
nano	<b>n</b>	$10^{-9}$	<code>si_nano</code>
pico	<b>p</b>	$10^{-12}$	<code>si_pico</code>
femto	<b>f</b>	$10^{-15}$	<code>si_femto</code>
atto	<b>a</b>	$10^{-18}$	<code>si_atto</code>
zepto	<b>z</b>	$10^{-21}$	<code>si_zepto</code>
yocto	<b>y</b>	$10^{-24}$	<code>si_yocto</code>
ronto	<b>r</b>	$10^{-27}$	<code>si_ronto</code>
quecto	<b>q</b>	$10^{-30}$	<code>si_quecto</code>

**Encoding note:** **μ** is U+00B5 (MICRO SIGN), UTF-8: `0xC2 0xB5`. U+03BC (GREEK SMALL LETTER MU) is a distinct code point and is **not** accepted.

## Prefix-Symbol Ambiguities

Several prefix symbols overlap with base unit symbols. The **~** separator is the disambiguator: **m~** introduces the milli prefix; bare **m** is the meter.

Symbol	As prefix	As base unit
<b>m</b>	milli	meter
<b>d</b>	deci	day
<b>h</b>	hecto	hour
<b>T</b>	tera	tesla
<b>f</b>	femto	farad
<b>a</b>	atto	(none)
<b>S</b>	(none)	siemens

**d~s** = decisecond; **d** alone = day.



## 4.2 IEC Binary Prefixes

IEC 80000-13 binary prefixes are used for digital quantities (`b` and `B` only).

Prefix	Symbol	Factor	Enum value
kibi	<code>Ki</code>	$2^{10}$	<code>iec_kibi</code>
mebi	<code>Mi</code>	$2^{20}$	<code>iec_mebi</code>
gibi	<code>Gi</code>	$2^{30}$	<code>iec_gibi</code>
tebi	<code>Ti</code>	$2^{40}$	<code>iec_tebi</code>
pebi	<code>Pi</code>	$2^{50}$	<code>iec_pebi</code>
exbi	<code>Ei</code>	$2^{60}$	<code>iec_exbi</code>
zebi	<code>Zi</code>	$2^{70}$	<code>iec_zebi</code>
yobi	<code>Yi</code>	$2^{80}$	<code>iec_yobi</code>
robi	<code>Ri</code>	$2^{90}$	<code>iec_robi</code>
quebi	<code>Qi</code>	$2^{100}$	<code>iec_quebi</code>

### Prefix-Unit Validity Constraints

- **IEC prefixes** (`Ki`...`Qi`) are only permitted on `b` and `B`. `Ki~m` → `error_unit_illegal`.
- **SI sub-kilo prefixes** (`d`, `c`, `m`, `μ`, `n`, `p`, `f`, `a`, `z`, `y`, `r`, `q`, `da`, `h`) are forbidden on `b` and `B`.
- **German units** (`bu_pfund` through `bu_scheffel`) accept only `si_none` / `iec_none`.
- **Currency units** accept SI prefixes of any magnitude (see §9.4). IEC prefixes are forbidden on all currency codes.

```
.valid1   = <uint:64,Ki~B> 8;      # OK: IEC prefix on byte
.valid2   = <uint:32,M~b> 100;    # OK: SI mega on bit
.valid3   = <float_dec:64,k~$USD> 250.0; # OK: kilo on currency
.invalid1 = <uint:64,Ki~m> 1;      # ERROR: IEC prefix on meter
.invalid2 = <uint:32,m~B> 512;    # ERROR: SI milli on byte
.invalid3 = <float_dec:64,Ki~$USD> 1; # ERROR: IEC prefix on currency
```

## 5. Unit Notation Grammar

### 5.1 Simple Units

```
unit-component = [ prefix "~" ] base-unit [ unit-exponent ]
```

```
.temperature = <float:64,K>      300.0;  # kelvin
.distance    = <float:64,k~m>    1.5;    # kilometer
.frequency   = <float:64,M~Hz>   2400;   # megahertz
.storage     = <uint:64,Ki~B>    1024;   # kibibytes
.fund_nav    = <float_dec:64,k~$USD> 250.0; # $250,000
```

## 5.2 Compound Units

```
compound-unit = "no_unit"
               | unit-component { unit-sep unit-component }

unit-sep      = "*" | "/" | "."          (* · = U+00B7 MIDDLE DOT *)

unit-component = [ prefix "~" ] base-unit [ unit-exponent ]

unit-exponent  = [ exp-sign ] exp-digit
               | "^" [ "-" | "+" ] ASCII-digit

exp-sign       = "+" | "-"

exp-digit      = "1" | "2" | "3" | "4" | "5"
               | "6" | "7" | "8" | "9"
```

Currency codes participate in compound expressions using the same separators:

```
.gold_spot    = <float_dec:64,$USD/oz_t> 2351.40; # $/troy oz
.rent         = <float_dec:64,$EUR/m²>   12.50; # €/m²
.billing_rate = <float_dec:64,$EUR/h>    150.00; # €/h
.eur_usd      = <float_dec:64,$USD/$EUR> 1.0842; # exchange rate
```

The sub-grammar is **semantic**, not lexical. The outer lexer captures the type-annotation body as a raw byte sequence; `bvn_parse_unit` parses the unit string portion after the lexer finishes.

## 5.3 Separators

Character	Code point	UTF-8 bytes	Meaning
*	U+002A	0x2A	Multiplication
·	U+00B7	0xC2 0xB7	Multiplication (preferred visual form)
/	U+002F	0x2F	Division

· and \* are semantically identical and may be mixed freely.

## 5.4 Denominator Semantics

The first / sets a latching "in-denominator" flag to `true` for all subsequent components. Additional / separators do not toggle back to the numerator. `k~g·m/s²` stores:

```
[0]: gram,    exp_linear,    si_kilo  ← numerator
[1]: meter,   exp_linear,    si_none  ← numerator
[2]: second,  exp_neg_square, si_none  ← denominator (exponent negated)
```

Both  $\text{k}\sim\text{g}\cdot\text{m}/\text{s}^2$  and  $\text{k}\sim\text{g}\cdot\text{m}\cdot\text{s}^{-2}$  parse to identical `value_unit_t` representations.

## Parenthesised grouping

A `(...)` group is a sub-expression evaluated independently. Like any factor it obeys the latching denominator, so a `/` before a group negates the group's net component exponents as a whole. This makes the readable denominator form work and compose correctly:

```
.pressure = <float:64,k~g/(m·s²)> 101325; # kg·m⁻¹·s⁻² – same as k~g/m·s²
.areal    = <float:64,(k~g/m)·s²> 1.0;    # kg·m⁻¹·s² – grouping changes the s sign
.rate     = <float:64,m/(s·s)> 1.0;       # m·s⁻²
```

$\text{k}\sim\text{g}/(\text{m}\cdot\text{s}^2)$  and  $\text{k}\sim\text{g}/\text{m}\cdot\text{s}^2$  therefore parse to the **same** `value_unit_t` (the canonical, parenless form is what the writer emits). Rules: an explicit separator is required before a group (`m·(s)`, not `m(s)`); a group is not followed by its own exponent (`(m·s)²` is rejected — write `m²·s²`); parentheses must balance; empty groups `( )` are rejected; nesting is bounded at 16. Unmatched or malformed groups raise `error_unit_illegal`.

## 6. Exponents

Exponents are limited to integer values in the range **−9 ... +9**.

### 6.1 Unicode Superscript Form

Glyph	Code point	UTF-8 bytes	Maps to
1	U+00B9	0xC2 0xB9	<code>exp_linear</code>
2	U+00B2	0xC2 0xB2	<code>exp_square</code>
3	U+00B3	0xC2 0xB3	<code>exp_cubic</code>
4	U+2074	0xE2 0x81 0xB4	<code>exp_quartic</code>
5	U+2075	0xE2 0x81 0xB5	<code>exp_quintic</code>
6	U+2076	0xE2 0x81 0xB6	<code>exp_sextic</code>
7	U+2077	0xE2 0x81 0xB7	<code>exp_septic</code>
8	U+2078	0xE2 0x81 0xB8	<code>exp_octic</code>
9	U+2079	0xE2 0x81 0xB9	<code>exp_nonic</code>
+	U+207A	0xE2 0x81 0xBA	positive sign (no-op)
−	U+207B	0xE2 0x81 0xBB	negate exponent

## 6.2 ASCII Caret Form

ASCII form	Equivalent Unicode	Parsed as
<code>m^2</code>	<code>m²</code>	<code>exp_square</code>
<code>s^-2</code>	<code>s<sup>-2</sup></code>	<code>exp_neg_square</code>
<code>m^+2</code>	<code>m²</code>	<code>exp_square</code>
<code>kg^1</code>	<code>kg¹</code>	<code>exp_linear</code>

Only a **single ASCII digit** is permitted after the caret.

## 6.3 Exponent Edge Cases

- **`exp_invalid` (value 0):** Zero-initialized sentinel. API functions that have an error output path reject it and signal an error. The two prefix query functions (`bvnr_unit_prefix_factor`, `bvnr_unit_prefix_exponent`) silently skip components with `exp_invalid`.
- **`exp_linear` (value 1):** Stored for both an explicit `¹`/`^1` and any component written without an exponent suffix.

## 7. The `no_unit` Keyword

The literal `no_unit` declares a value as **explicitly dimensionless**:

```
.ratio      = <float:64,no_unit> 0.95;
.count      = <uint:32,no_unit> 1000;
```

`bvnr_parse_unit` detects `no_unit` via `memcmp` and returns `BVNR_UNIT_NONE` (`num_components = 0`).

**Omitting the unit parameter** (e.g. `<float:64>`) yields `BVNR_UNIT_NO_PREFIX(bu_none)` (`num_components = 1`, `base == bu_none`). Both forms are semantically equivalent — they compare as compatible via `bvnr_units_compatible` and both serialize to `"no_unit"` — but they are structurally distinct internal states.

## 8. Constraints and Limits

Constraint	Value	Error on violation
Maximum components per compound unit	8 ( <code>BVNR_MAX_UNIT_COMPONENTS</code> )	<code>error_unit_illegal</code>
Empty component between separators (e.g. <code>m//s</code> )	Not allowed	<code>error_unit_illegal</code>
Maximum raw unit string length	Enforced by type-buffer limit	<code>error_unit_too_long</code>

Constraint	Value	Error on violation
Null or empty unit string	Rejected by <code>bvn_parse_unit</code>	<code>ok = false</code>

`a/b/c` parses as `a / (b·c)` → components `[a, b-1, c-1]`. The "no toggle back" rule means `/` always adds to the denominator; it never re-enters the numerator.

## 9. Currency Codes

Currency amounts are dimensional quantities in financial computing. `$19.99 USD` carries a denomination dimension just as `9.81 m/s2` carries an acceleration dimension. Bovnar extends the unit system with 214 currency and cryptocurrency codes so that monetary data can be annotated and round-tripped with the same precision guarantees as physical measurements.

### 9.1 The `$` Sigil Rule

As of spec 1.0 a currency is recognised **only** in its `$`-sigil form (`$USD`, `$BTC`, or prefixed `k~$EUR`). The sigil — and nothing else — dispatches a component to the currency table; see §10.4 for the full rules and rationale.

Classification happens at the lookup stage, per unit component:

1. A component introduced by `$` (after any SI/IEC prefix and its `~`) is looked up in the **currency table**. If the code is not found there, `error_unit_illegal` is raised — a `$` introduces a currency and nothing else.
2. A component **without** a `$` is looked up only in the **physical unit table**. A bare code such as `USD` or `CUP` is therefore never a currency; if it is not a physical unit it is `error_unit_illegal`.

Because the sigil is mandatory, a bare uppercase code can never collide with a physical-unit symbol — present or future — so the two namespaces are disjoint by construction. The collision cases this resolves are catalogued for reference in §10.

### 9.2 ISO 4217 Fiat Currencies and Precious Metals

164 ISO 4217 alphabetic codes are supported, including precious-metal X-codes. They occupy the `value_base_unit_t` slot range **134 ... 297** alphabetically (AED first, ZWL last), but — unlike physical units — they have **no named `bu_*` enumerators**: a currency is resolved from its `$`-sigil code by `bvn_parse_currency_str` and carried as the numeric `base` value (the currency catalogue in `bovnar_currency.c` is index-aligned to these slots). Two codes are historical and retained for compatibility: `HRK` (Croatian Kuna, retired 2023-01-01 when Croatia adopted the Euro) and `SLL` (Sierra Leonean Leone (old), replaced by `SLE` in 2022).

The `minor_unit` field carries the exponent N such that 1 major unit =  $10^N$  minor units (e.g. 1 USD = 100 cents, N=2). Applications reading integer-annotated values (e.g. `<uint:64,$KWD>`) should call `bvn_currency_minor_unit` to determine the correct decimal shift. Minor units are **bold** below when they differ from 2. `Num` is the ISO 4217 numeric identifier.

Code	Num	Min	Name
AED	784	2	UAE Dirham
AFN	971	2	Afghan Afghani
ALL	8	2	Albanian Lek
AMD	51	2	Armenian Dram
ANG	532	2	Netherlands Antillean Guilder
AOA	973	2	Angolan Kwanza
ARS	32	2	Argentine Peso
AUD	36	2	Australian Dollar
AWG	533	2	Aruban Florin
AZN	944	2	Azerbaijani Manat
BAM	977	2	Bosnia-Herzegovina Convertible Mark
BBD	52	2	Barbados Dollar
BDT	50	2	Bangladeshi Taka
BGN	975	2	Bulgarian Lev
BHD	48	<b>3</b>	Bahraini Dinar
BIF	108	<b>0</b>	Burundian Franc
BMD	60	2	Bermudian Dollar
BND	96	2	Brunei Dollar
BOB	68	2	Boliviano
BRL	986	2	Brazilian Real
BSD	44	2	Bahamian Dollar
BTN	64	2	Bhutanese Ngultrum
BWP	72	2	Botswana Pula
BYN	933	2	Belarusian Ruble
BZD	84	2	Belize Dollar
CAD	124	2	Canadian Dollar
CDF	976	2	Congolese Franc
CHF	756	2	Swiss Franc
CLF	990	<b>4</b>	Unidad de Fomento
CLP	152	<b>0</b>	Chilean Peso

Code	Num	Min	Name
CNY	156	2	Chinese Yuan
COP	170	2	Colombian Peso
CRC	188	2	Costa Rican Colon
CUP	192	2	Cuban Peso
CVE	132	2	Cape Verdean Escudo
CZK	203	2	Czech Koruna
DJF	262	0	Djiboutian Franc
DKK	208	2	Danish Krone
DOP	214	2	Dominican Peso
DZD	12	2	Algerian Dinar
EGP	818	2	Egyptian Pound
ERN	232	2	Eritrean Nakfa
ETB	230	2	Ethiopian Birr
EUR	978	2	Euro
FJD	242	2	Fijian Dollar
FKP	238	2	Falkland Islands Pound
GBP	826	2	Pound Sterling
GEL	981	2	Georgian Lari
GHS	936	2	Ghanaian Cedi
GIP	292	2	Gibraltar Pound
GMD	270	2	Gambian Dalasi
GNF	324	0	Guinean Franc
GTQ	320	2	Guatemalan Quetzal
GYD	328	2	Guyanese Dollar
HKD	344	2	Hong Kong Dollar
HNL	340	2	Honduran Lempira
HRK	191	2	Croatian Kuna ( <i>historical; retired 2023-01-01</i> )
HTG	332	2	Haitian Gourde
HUF	348	2	Hungarian Forint
IDR	360	2	Indonesian Rupiah
ILS	376	2	Israeli New Shekel
INR	356	2	Indian Rupee
IQD	368	3	Iraqi Dinar
IRR	364	2	Iranian Rial

Code	Num	Min	Name
ISK	352	0	Icelandic Krona
JMD	388	2	Jamaican Dollar
JOD	400	3	Jordanian Dinar
JPY	392	0	Japanese Yen
KES	404	2	Kenyan Shilling
KGS	417	2	Kyrgyzstani Som
KHR	116	2	Cambodian Riel
KMF	174	0	Comorian Franc
KPW	408	2	North Korean Won
KRW	410	0	South Korean Won
KWD	414	3	Kuwaiti Dinar
KYD	136	2	Cayman Islands Dollar
KZT	398	2	Kazakhstani Tenge
LAK	418	2	Laotian Kip
LBP	422	2	Lebanese Pound
LKR	144	2	Sri Lankan Rupee
LRD	430	2	Liberian Dollar
LSL	426	2	Lesotho Loti
LYD	434	3	Libyan Dinar
MAD	504	2	Moroccan Dirham
MDL	498	2	Moldovan Leu
MGA	969	2	Malagasy Ariary
MKD	807	2	Macedonian Denar
MMK	104	2	Myanmar Kyat
MNT	496	2	Mongolian Togrog
MOP	446	2	Macanese Pataca
MRU	929	2	Mauritanian Ouguiya
MUR	480	2	Mauritian Rupee
MVR	462	2	Maldivian Rufiyaa
MWK	454	2	Malawian Kwacha
MXN	484	2	Mexican Peso
MYR	458	2	Malaysian Ringgit
MZN	943	2	Mozambican Metical
NAD	516	2	Namibian Dollar



Code	Num	Min	Name
NGN	566	2	Nigerian Naira
NIO	558	2	Nicaraguan Cordoba
NOK	578	2	Norwegian Krone
NPR	524	2	Nepalese Rupee
NZD	554	2	New Zealand Dollar
OMR	512	3	Omani Rial
PAB	590	2	Panamanian Balboa
PEN	604	2	Peruvian Sol
PGK	598	2	Papua New Guinean Kina
PHP	608	2	Philippine Peso
PKR	586	2	Pakistani Rupee
PLN	985	2	Polish Zloty
PYG	600	0	Paraguayan Guarani
QAR	634	2	Qatari Riyal
RON	946	2	Romanian Leu
RSD	941	2	Serbian Dinar
RUB	643	2	Russian Ruble
RWF	646	0	Rwandan Franc
SAR	682	2	Saudi Riyal
SBD	90	2	Solomon Islands Dollar
SCR	690	2	Seychellois Rupee
SDG	938	2	Sudanese Pound
SEK	752	2	Swedish Krona
SGD	702	2	Singapore Dollar
SHP	654	2	Saint Helena Pound
SLE	925	2	Sierra Leonean Leone
SLL	694	2	Sierra Leonean Leone (old) <i>(historical; replaced by SLE 2022)</i>
SOS	706	2	Somali Shilling
SSP	728	2	South Sudanese Pound
SRD	968	2	Surinamese Dollar
STN	930	2	Sao Tome and Principe Dobra
SVC	222	2	Salvadoran Colon
SYP	760	2	Syrian Pound
SZL	748	2	Swazi Lilangeni

Code	Num	Min	Name
THB	764	2	Thai Baht
TJS	972	2	Tajikistani Somoni
TMT	934	2	Turkmenistan Manat
TND	788	3	Tunisian Dinar
TOP	776	2	Tongan Pa'anga
TRY	949	2	Turkish Lira
TTD	780	2	Trinidad and Tobago Dollar
TWD	901	2	New Taiwan Dollar
TZS	834	2	Tanzanian Shilling
UAH	980	2	Ukrainian Hryvnia
UGX	800	0	Ugandan Shilling
USD	840	2	US Dollar
UYU	858	2	Uruguayan Peso
UZS	860	2	Uzbekistani Som
VES	928	2	Venezuelan Bolivar Soberano
VND	704	0	Vietnamese Dong
VUV	548	0	Vanuatu Vatu
WST	882	2	Samoan Tala
XAF	950	0	CFA Franc BEAC
XAG	961	0	Silver
XAU	959	0	Gold
XCD	951	2	East Caribbean Dollar
XDR	960	0	Special Drawing Rights
XOF	952	0	CFA Franc BCEAO
XPD	964	0	Palladium
XPF	953	0	CFP Franc
XPT	962	0	Platinum
XTS	963	0	Test currency (ISO 4217 reserved; do not use in production)
YER	886	2	Yemeni Rial
ZAR	710	2	South African Rand
ZMW	967	2	Zambian Kwacha
ZWL	932	2	Zimbabwean Dollar

`CLF` (Unidad de Fomento) is the only currency with 4 minor units. The two historical codes `HRK` and `SLL` are retained for compatibility but should not be used for new data.

## 9.3 Cryptocurrencies

50 cryptocurrencies are supported, with 3- or 4-letter uppercase tickers. They occupy `value_base_unit_t` slots **298 ... 347** and, like the fiat codes, have **no named `bu_*` enumerators** — they are resolved by `bnv_parse_currency_str` and carried as the numeric `base` value. The `minor_unit` field holds the canonical on-chain decimal places. `numeric_code = 0` for all cryptocurrencies.

**Min** = `minor_unit` = on-chain decimal places. E.g. `<uint:64,$BTC>` stores satoshis; divide by  $10^8$  to obtain whole BTC.

Code	Min	Subunit	Name
<code>BTC</code>	8	satoshi	Bitcoin
<code>ETH</code>	18	wei	Ethereum
<code>SOL</code>	9	lamport	Solana
<code>XRP</code>	6	drop	XRP
<code>BNB</code>	18	—	BNB
<code>ADA</code>	6	lovelace	Cardano
<code>LTC</code>	8	—	Litecoin
<code>DOT</code>	10	planck	Polkadot
<code>XMR</code>	12	piconero	Monero
<code>ETC</code>	18	—	Ethereum Classic
<code>BCH</code>	8	—	Bitcoin Cash
<code>XLM</code>	7	stroop	Stellar
<code>FIL</code>	18	—	Filecoin
<code>ICP</code>	8	—	Internet Computer
<code>TRX</code>	6	—	TRON
<code>EOS</code>	4	—	EOS
<code>VET</code>	18	—	VeChain
<code>NEO</code>	8	—	Neo
<code>ZEC</code>	8	—	Zcash
<code>UNI</code>	18	—	Uniswap
<code>ARB</code>	18	—	Arbitrum

Code	Min	Subunit	Name
SUI	9	—	Sui
TON	9	—	Toncoin
INJ	18	—	Injective
SEI	6	—	Sei
APT	8	—	Aptos
TAO	9	—	Bittensor
WIF	6	—	dogwifhat
DOGE	8	koinu	Dogecoin
LINK	18	—	Chainlink
USDT	6	—	Tether
USDC	6	—	USD Coin
AVAX	18	—	Avalanche
ATOM	6	—	Cosmos
POL	18	—	Polygon
NEAR	24	—	NEAR Protocol
ALGO	6	—	Algorand
HBAR	8	—	Hedera
AAVE	18	—	Aave
MKR	18	—	Maker
DAI	18	—	Dai
STX	6	—	Stacks
GRT	18	—	The Graph
LDO	18	—	Lido DAO
BONK	5	—	Bonk
PEPE	18	—	Pepe
SHIB	18	—	Shiba Inu
JUP	6	—	Jupiter
PYTH	6	—	Pyth Network
RUNE	8	—	THORChain

## 9.4 Prefix Rules for Currency Units

**All SI prefixes** are permitted on all currency units. `k~USD` denotes "values in thousands of USD" — a common scale annotation in financial reporting.

```
.fund_nav = <float_dec:64,k~$USD> 250.0; # $250,000
.gdp      = <float_dec:64,M~$EUR> 42800.0; # €42.8 billion
.eth_gwei = <float_dec:64,G~$ETH> 35.0; # 35 Gwei gas price
```

**IEC binary prefixes** (`Ki~`, `Mi~`, ...) are **forbidden** on all currency units.

`bvn_currency_prefix_valid()` returns `false` for any IEC prefix; the parser raises `error_unit_illegal`.

## 9.5 Compound Currency Expressions

Currency codes participate in compound unit expressions using the existing separators:

```
.gold_spot = <float_dec:64,$USD/oz_t> 2351.40; # $/troy oz
.wheat     = <float_dec:64,$USD/bsh> 5.82; # $/bushel
.rent      = <float_dec:64,$EUR/m²> 12.50; # €/m²
.billing_rate = <float_dec:64,$EUR/h> 150.00; # €/h
.eur_usd     = <float_dec:64,$USD/$EUR> 1.0842; # exchange rate
.eth_btc     = <float_dec:64,$BTC/$ETH> 0.05610; # cross-crypto rate
```

Currency × currency compounds (`USD·EUR`) are syntactically valid and produce no error. Their financial interpretation is the application's responsibility.

### Exchange Rate Timestamps

Bovnar annotates denomination; it does not store exchange rates or timestamps. The timestamp belongs in a separate field:

```
.snapshot = {
  .epoch = <uint:64,s> 1716400000;
  .eur_usd = <float_dec:64,$USD/$EUR> 1.0842;
};
```

## 9.6 Compatibility Rules

`bvn_units_compatible()` requires no modification for currencies. Two currency expressions are structurally compatible only if they have identical component sequences including base enum values. Since `bu_usd ≠ bu_eur`, `USD` and `EUR` are already structurally incompatible under the existing comparison logic.

## 9.7 Type Pairing Recommendations

Use case	Recommended annotation	Rationale
Decimal monetary amount	<code>&lt;float_dec:64,\$USD&gt;</code>	Exact decimal; 16 significant digits
High-precision / actuarial	<code>&lt;float_dec:128,\$USD&gt;</code>	34 significant digits
Integer minor-unit storage	<code>&lt;uint:64,\$USD&gt;</code>	Value in cents; app reads <code>minor_unit()</code>
Negative balances in minor units	<code>&lt;sint:64,\$USD&gt;</code>	

Use case	Recommended annotation	Rationale
Zero-minor-unit currency	<code>&lt;uint:64,\$JPY&gt;</code>	Integer is the only correct representation
3-minor-unit currency	<code>&lt;uint:64,\$KWD&gt;</code>	Value in fils
Commodity price	<code>&lt;float_dec:64,\$USD/oz_t&gt;</code>	\$/troy oz
Exchange rate	<code>&lt;float_dec:64,\$USD/\$EUR&gt;</code>	USD per EUR
On-chain satoshi balance	<code>&lt;uint:64,\$BTC&gt;</code>	Integer satoshis
Human-readable BTC amount	<code>&lt;float_dec:64,\$BTC&gt;</code>	

**float (binary floating-point) is discouraged** for monetary amounts. Binary fractions cannot represent 0.10 USD exactly. Use `float_dec` for decimal-exact storage.

**float\_fix is wrong** for monetary values. Q-format stores values as `integer × 2(-N)` — a binary fractional resolution. No power of 2 equals a power of 10 (except  $2^0 = 10^0 = 1$ ), so no Q value exactly represents cents.

## 10. Symbol Disambiguation

This section documents how a physical-unit token and a currency token are kept apart. The mandatory `$` currency sigil (§10.4) is the normative rule and resolves every potential collision; the look-alike tables that follow are retained for reference.

### 10.1 The Namespace Rule as Disambiguator

The mandatory `$` sigil (§9.1, normative rule in §10.4) makes the two namespaces disjoint by construction: the sigil — and nothing else — selects the currency table, so a bare token is **always** a physical-unit lookup and can never be mistaken for a currency.

Written token	Looked up in	Result
<code>\$USD</code> , <code>\$BTC</code> , <code>k~\$EUR</code>	currency table (sigil present)	currency, or <code>error_unit_illegal</code> if the code is unknown
<code>m</code> , <code>Hz</code> , <code>cup</code> , <code>BTU</code> , <code>k~g</code> (no <code>\$</code> )	physical unit table only	physical unit, or <code>error_unit_illegal</code> if unknown
<code>USD</code> , <code>CUP</code> , <code>XYZ</code> (no <code>\$</code> )	physical unit table only	<code>error_unit_illegal</code> — not physical units, and a bare code is never a currency

Because the lookup is sigil-driven rather than spelling-driven, **case is no longer load-bearing for disambiguation**: `cup` and `CUP` are both physical-unit lookups (the first matches `bu_cup`, the second is `error_unit_illegal`), and the Cuban Peso is written `$CUP`.

## 10.2 Exhaustive Conflict Table

Before the sigil, several uppercase tokens *looked* like they could be either a physical unit or a currency. The sigil removes the ambiguity outright; the table below is retained as a reference for the codes that previously needed disambiguation. In every row, the bare form is a physical-unit lookup and the currency is only ever the `$`-prefixed form.

Token	Bare form (no <code>\$</code> )	<code>\$</code> -sigil form	Notes
<code>cup</code> / <code>CUP</code>	<code>cup</code> → US cup ( <code>bu_cup</code> ); <code>CUP</code> → <code>error_unit_illegal</code>	<code>\$CUP</code> → Cuban Peso (ISO 4217:192)	the classic look-alike, now fully separated
<code>BTU</code>	<code>BTU</code> → International Table BTU ( <code>bu_btu</code> ); <code>Btu</code> , <code>btu</code> also accepted	( <i>not ISO 4217</i> )	uppercase <code>BTU</code> is a physical alias, not a currency
<code>SOL</code>	<code>SOL</code> → <code>error_unit_illegal</code> (no physical unit)	<code>\$SOL</code> → Solana (crypto)	
<code>BAR</code>	<code>BAR</code> → <code>error_unit_illegal</code> ; use lowercase <code>bar</code>	( <i>not ISO 4217</i> )	
<code>ERG</code>	<code>ERG</code> → <code>error_unit_illegal</code> ; use lowercase <code>erg</code>	( <i>not ISO 4217</i> )	
<code>CAD</code> , <code>AUD</code> , <code>GBP</code> , <code>XAU</code>	<code>error_unit_illegal</code> (no physical unit)	<code>\$CAD</code> , <code>\$AUD</code> , <code>\$GBP</code> , <code>\$XAU</code> → the respective currencies	

**Key finding:** with the sigil, no token is simultaneously a valid bare physical-unit symbol and a valid currency — currencies live entirely under `$`, physical units entirely without it.

## 10.3 The CUP Case in Detail

`CUP` is the classic example because the ISO 4217 code for the Cuban Peso shares its letters with the English word for the culinary measure. Under the sigil rule the two are unambiguous:

Written in BVNR	Resolved as	Enum value	SI factor
<code>cup</code>	US cup	<code>bu_cup</code>	$2.365882365 \times 10^{-4} \text{ m}^3$
<code>cups</code>	US cup (long form)	<code>bu_cup</code>	$2.365882365 \times 10^{-4} \text{ m}^3$
<code>CUP</code>	( <i>error</i> ) — bare uppercase is not a physical unit	—	<code>error_unit_illegal</code>
<code>\$CUP</code>	Cuban Peso	<code>bu_cup</code> ( <i>currency enum</i> )	— (monetary, no SI factor)

```
.recipe_volume = <float_dec:32,cup> 2.0;    # 2 US cups (volume)
.balance       = <float_dec:64,$CUP> 15.00;  # 15 Cuban Pesos (currency)
# .bad        = <float_dec:64,CUP> 15.00;    # error_unit_illegal: bare 'CUP' is not a unit
```

Calling `bvn_unit_to_si_factor` on a `$CUP` unit returns `*ok = false` because `bvn_find_si_conv` skips currency enum values. Calling `bvn_unit_is_currency` on a `cup` unit returns `false`. The two are completely disjoint in both parsing and the conversion API — and, because the peso *must* be written `$CUP`, a user who writes the bare `CUP` intending the culinary cup gets an immediate `error_unit_illegal` (the correct spelling is lowercase `cup`) rather than a silently-accepted currency.

## 10.4 The Mandatory Currency Sigil

A currency code carries a **mandatory \$ sigil** as of spec 1.0. This is the resolution of the `CUP` (Cuban Peso) vs `cup` (the physical cup) namespace collision: a currency is *only* recognised in its sigil form, so a bare code can never collide with a physical-unit symbol — present or future.

```
# Currencies – the '$' sigil is required:
.price   = <float_dec:64,$USD>      19.99;    # US Dollar
.btc      = <uint:64,$BTC>           54782000;  # Bitcoin (satoshis)
.fund     = <float_dec:64,k~$EUR>    250.0;     # kilo-Euro (prefix before the sigil)
.spot     = <float_dec:64,$USD/oz_t> 2351.40;   # currency / physical-unit compound

# A bare code is no longer a currency:
.volume   = <float_dec:32,cup>       2.0;       # the physical 'cup', unambiguously
# .bad    = <float_dec:64,USD> 1.0;           # error_unit_illegal: bare 'USD' is not a
unit (needs '$')
```

The sigil attaches directly before the currency code, after any SI/IEC prefix and its `~` (`k~$EUR`, `M~$USD`). It is accepted in inline units and type annotations alike, and the writer emits it on output so values round-trip. Because this **breaks** any document that used bare currency codes, it was made before the 1.0 freeze — afterwards it would be an incompatible change (see the Versioning & Stability section of the specification).

## 11. C Data Model

### 11.1 Enumerations

`prefix_system_t`

```
typedef enum prefix_system_e {
    prefix_si,      /* SI decimal prefixes (or no prefix) */
    prefix_iec      /* IEC binary prefixes */
} prefix_system_t;
```



**si\_prefix\_id\_t**

```
typedef enum si_prefix_id_e {
    si_none = 0,
    si_zepto, si_ronto, si_yocto, si_zepto, si_atto,
    si_femto, si_pico, si_nano, si_micro, si_milli,
    si_cent, si_deci,
    si_deca, si_hecto, si_kilo, si_mega, si_giga,
    si_tera, si_peta, si_exa, si_zetta, si_yotta,
    si_ronna, si_quetta
} si_prefix_id_t;
```

**iec\_prefix\_id\_t**

```
typedef enum iec_prefix_id_e {
    iec_none = 0,
    iec_kibi, iec_mebi, iec_gibi, iec_tebi, iec_pebi,
    iec_exbi, iec_zebi, iec_yobi, iec_robi, iec_quebi
} iec_prefix_id_t;
```

**value\_base\_unit\_t**

Non-German physical units occupy positions 1–133 ( `bu_bit` ... `bu_bushel` ). Currency codes occupy positions 134–347 — an unnamed slot range (no `bu_*` enumerators; see §9.2/§9.3). German physical units are appended after the entire currency range at positions 348–360 ( `bu_pfund` ... `bu_scheffel` ). Additional physical units occupy positions 361–367 ( `bu_survey_foot` ... `bu_baud` ), historical temperature scales 368–371 ( `bu_delisle` ... `bu_romer` ), and dimensionless ratio units 372–377 ( `bu_percent` ... `bu_ppb` ). `bvn_unit_is_currency(base)` returns `true` for any value in the range 134–347.

```

typedef enum value_base_unit_e {
    bu_none = 0,                /* dimensionless / no unit */

    /* Digital */
    bu_bit, bu_byte,

    /* SI base */
    bu_second, bu_meter, bu_gram, bu_ampere, bu_kelvin,
    bu_mol, bu_candela,

    /* Named SI-derived */
    bu_hertz, bu_newton, bu_pascal, bu_joule, bu_watt,
    bu_volt, bu_ohm, bu_farad, bu_coulomb, bu_siemens,
    bu_weber, bu_tesla, bu_henry, bu_lumen, bu_lux,
    bu_becquerel, bu_gray, bu_sievert, bu_katal,
    bu_liter, bu_minute, bu_hour, bu_day, bu_degree, bu_celsius,
    bu_radian, bu_steradian,
    bu_tonne, bu_bar,
    bu_electronvolt, bu_dalton, bu_astronomical_unit,
    bu_hectare, bu_week, bu_year,

    /* Imperial/US - length */
    bu_inch, bu_foot, bu_yard, bu_mile, bu_nautical_mile,
    bu_angstrom, bu_light_year, bu_parsec, bu_furlong, bu_fathom,

    /* Imperial/US - mass */
    bu_pound, bu_ounce, bu_grain, bu_stone, bu_short_ton,
    bu_long_ton, bu_troy_ounce, bu_carat,

    /* Temperature */
    bu_fahrenheit,

    /* Pressure */
    bu_atmosphere, bu_mmhg, bu_torr, bu_psi,

    /* Energy */
    bu_calorie, bu_btu, bu_erg, bu_therm,

    /* Power */
    bu_horsepower,

    /* Force */
    bu_pound_force, bu_dyne, bu_kip,

    /* Speed */
    bu_knot,

    /* Volume - US */
    bu_gallon, bu_gallon_uk, bu_quart, bu_pint, bu_cup,
    bu_fluid_ounce, bu_tablespoon, bu_teaspoon, bu_barrel,

    /* Area */
    bu_acre, bu_barn,

    /* Angle */
    bu_arcminute, bu_arcsecond, bu_grad,

    /* CGS */
    bu_poise, bu_stokes, bu_gauss, bu_maxwell, bu_oersted,
    bu_stilb, bu_phot, bu_galileo,

    /* Radiation */
    bu_curie, bu_roentgen, bu_rem,

```

```

/* Logarithmic */
bu_neper, bu_decibel,

bu_rankine,          /* Temperature – absolute Fahrenheit scale */
bu_slug,             /* Imperial mass */
bu_thou,             /* Imperial length */

/* Volume – UK imperial */
bu_pint_uk, bu_fluid_ounce_uk, bu_quart_uk,

/* Electrical power */
bu_var, bu_volt_ampere,

/* Force (additional) */
bu_kilogram_force,

/* Pressure (additional) */
bu_inch_hg,

/* Rotational frequency */
bu_rpm,

/* Energy (additional) */
bu_foot_pound,

/* Mass (additional) */
bu_dram, bu_pennyweight,

/* Length (additional) */
bu_chain, bu_rod,

/* Volume (additional) */
bu_gill, bu_gill_uk,

/* Acceleration / gravity */
bu_standard_gravity,

/* Power (additional) */
bu_metric_horsepower,

/* Angle (additional) */
bu_revolution,

/* Time (additional) */
bu_month, bu_fortnight,

/* Pressure (additional) */
bu_atmosphere_technical,

/* Textile */
bu_tex, bu_denier,

/* Apothecary / dry volume */
bu_fluid_dram, bu_minim, bu_peck, bu_bushel,

/* Slots 134–347 are the ISO 4217 fiat (134–297) and cryptocurrency
 * (298–347) ranges. These have NO named enumerators: the enum jumps
 * straight from bu_bushel (133) to bu_pfund (348). Currencies are
 * resolved by string via bvn_parse_currency_str and carried as the
 * numeric base value; the catalogue in bovnar_currency.c is index-
 * aligned to slots 134–347 (BVN_CURRENCY_FIAT_FIRST ... CRYPTO_LAST). */

/* Old German – placed after the currency range */
bu_pfund = 348, bu_zentner, bu_doppelzentner, bu_lot,
bu_prussian_line, bu_prussian_zoll, bu_prussian_fuss,

```

```

    bu_prussian_elle, bu_prussian_rute, bu_klafter,
    bu_german_mile, bu_morgen, bu_scheffel,  /* = 360 */

    /* Additional physical units */
    bu_survey_foot, bu_league, bu_cable, bu_hand,
    bu_quintal, bu_scruple, bu_baud,  /* = 367 */

    /* Historical temperature scales */
    bu_delisle, bu_newton_temp, bu_reaumur, bu_romer,  /* = 371 */

    /* Dimensionless ratio units */
    bu_percent, bu_per_mille, bu_per_myriad,
    bu_per_cent_mille, bu_ppm, bu_ppb,  /* = 377 */
} value_base_unit_t;

/* Total slot count – defined separately, not an enum member: */
/* #define BVN_VALUE_BASE_UNIT_COUNT 378  (bu_ppb + 1) */

```

### unit\_exponent\_t

```

typedef enum unit_exponent_e {
    exp_invalid    = 0,
    exp_linear     = 1, exp_square    = 2, exp_cubic     = 3,
    exp_quartic    = 4, exp_quintic   = 5, exp_sextic    = 6,
    exp_septic     = 7, exp_octic     = 8, exp_nonic     = 9,
    exp_neg_linear = -1, exp_neg_square=-2, exp_neg_cubic  = -3,
    exp_neg_quartic=-4, exp_neg_quintic=-5, exp_neg_sextic=-6,
    exp_neg_septic = -7, exp_neg_octic = -8, exp_neg_nonic = -9,
} unit_exponent_t;

```

`exp_invalid` (value `0`) is the zero-initialization sentinel. Always initialize components before use.

## 11.2 Structures

### value\_unit\_component\_t

```

typedef struct value_unit_component_s {
    value_base_unit_t  base;
    unit_exponent_t    exponent;
    value_unit_prefix_t prefix;
} value_unit_component_t;

```

where:

```

typedef struct value_unit_prefix_s {
    prefix_system_t system;
    union {
        si_prefix_id_t  si;
        iec_prefix_id_t iec;
    } id;
} value_unit_prefix_t;

```

**value\_unit\_t**

```
#define BVNR_MAX_UNIT_COMPONENTS 8

typedef struct value_unit_s {
    uint32_t          num_components;
    value_unit_component_t components[BVNR_MAX_UNIT_COMPONENTS];
} value_unit_t;
```

For an explicit `no_unit` annotation, `num_components == 0` (`BVN_UNIT_NONE`). For an absent unit parameter in an explicit annotation (e.g. `<float:64>`), `num_components == 1` with `base == bu_none`.

**bvnr\_data\_t (unit field)**

```
typedef struct bvnr_data_s {
    token_type_t      type;
    value_type_spec_t value_type;
    value_unit_t       value_unit; /* parsed physical unit or currency */
    const void*        data;
    uint32_t           length;
} bvnr_data_t;
```

## 11.3 Convenience Macros

```
/* Dimensionless or single-component without prefix */
#define BVN_UNIT_NO_PREFIX(b) \
    ((value_unit_t){ .num_components = 1, .components = {{ \
        .base=(b), .exponent=exp_linear, \
        .prefix.system=prefix_si, .prefix.id.si=si_none }}})

/* Single-component with SI prefix */
#define BVN_UNIT_SI(b, p) \
    ((value_unit_t){ .num_components = 1, .components = {{ \
        .base=(b), .exponent=exp_linear, \
        .prefix.system=prefix_si, .prefix.id.si=(p) }}})

/* Single-component with IEC prefix */
#define BVN_UNIT_IEC(b, p) \
    ((value_unit_t){ .num_components = 1, .components = {{ \
        .base=(b), .exponent=exp_linear, \
        .prefix.system=prefix_iec, .prefix.id.iec=(p) }}})

/* Single-component with SI prefix and explicit exponent */
#define BVN_UNIT_SI_EXP(b, p, e) \
    ((value_unit_t){ .num_components = 1, .components = {{ \
        .base=(b), .exponent=(e), \
        .prefix.system=prefix_si, .prefix.id.si=(p) }}})

/* Empty unit – explicit no_unit / internal sentinel */
#define BVN_UNIT_NONE ((value_unit_t){ .num_components = 0 })

/* Two-component compound unit, both SI-prefixed */
#define BVN_UNIT_COMPOUND2(b1,p1,e1, b2,p2,e2) \
    ((value_unit_t){ .num_components = 2, .components = { \
        { .base=(b1), .exponent=(e1), \
          .prefix.system=prefix_si, .prefix.id.si=(p1) }, \
        { .base=(b2), .exponent=(e2), \
          .prefix.system=prefix_si, .prefix.id.si=(p2) }}})
```

Usage:

```
value_unit_t kg = BVN_UNIT_SI(bu_gram, si_kilo);
value_unit_t gib = BVN_UNIT_IEC(bu_byte, iec_gibi);
value_unit_t m2 = BVN_UNIT_SI_EXP(bu_meter, si_none, exp_square);
value_unit_t usd = BVN_UNIT_NO_PREFIX(bu_usd); /* single currency unit */
value_unit_t none = BVN_UNIT_NONE;
```

## 12. C API Functions

### 12.1 Parsing a Unit String

```
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);
```

`bvn_parse_unit` parses a NUL-terminated UTF-8 unit string. `bvn_parse_unit_n` is the length-bounded variant used internally when the unit string is a slice of a larger type-annotation buffer. Both set `*ok = false` on any parse error (unknown prefix, unknown base unit, unknown currency code, too many components, empty component, or NULL input).

Single-pass parsing algorithm:

1. `memcmp` against `"no_unit"` → return `BVN_UNIT_NONE` immediately on match.
2. Scan for separator characters to distinguish simple vs. compound paths.
3. For compound units, split on `0x2A` (`*`), `0x2F` (`/`), `0xC2 0xB7` (`·`); parse each slice as a component; negate denominator exponents.
4. For each component, if it is introduced by the `$` sigil (after any prefix and its `~`), look the code up in the currency table; otherwise look it up in the physical unit table only. A bare code is never a currency.

```
bool ok;
value_unit_t u = bvn_parse_unit((const uint8_t*)"k~g·m/s²", &ok);
/* u.num_components == 3:
   [0]: bu_gram,    exp_linear,    si_kilo
   [1]: bu_meter,  exp_linear,    si_none
   [2]: bu_second, exp_neg_square, si_none */

value_unit_t c = bvn_parse_unit((const uint8_t*)"USD/oz_t", &ok);
/* c.num_components == 2:
   [0]: bu_usd,      exp_linear,    si_none (currency)
   [1]: bu_troy_ounce, exp_neg_linear, si_none */
```

### 12.2 Serializing a Unit

```
int32_t bvn_unit_to_string (value_unit_t u, char* buf, size_t bufsize);
int32_t bvn_unit_to_string_ex(value_unit_t u, char* buf, size_t bufsize,
                             bvn_unit_flags_t flags);
```

Serializes `u` to a canonical UTF-8 string. Returns the number of bytes written (excluding NUL) or `-1` on buffer overflow or invalid component.

Flag	Effect
<code>BVN_UNIT_FLAGS_NONE</code>	Unicode superscript exponents, no reduction
<code>BVN_UNIT_ASCII_EXP</code>	ASCII caret form ( <code>^N</code> ) for all exponents
<code>BVN_UNIT_REDUCE</code>	Reduce via <code>bvn_unit_reduce</code> before serializing

```
char buf[64];
value_unit_t u = /* k~g·m/s² */;
bvn_unit_to_string(u, buf, sizeof(buf));
/* buf == "k~g·m/s²" */

bvn_unit_to_string_ex(u, buf, sizeof(buf), BVN_UNIT_ASCII_EXP);
/* buf == "k~g*m/s^2" */
```

## Validation Predicate

```
bool bvn_unit_valid(value_unit_t u);
```

Returns `true` if every component has a valid exponent, known base unit (physical or currency), and a legal prefix for that base. Both serialization functions call this before writing.

## 12.3 Prefix Factor and Exponent Queries

```
double bvn_unit_prefix_factor (value_unit_t u);
int32_t bvn_unit_prefix_exponent(value_unit_t u);
```

`bvn_unit_prefix_factor` returns the multiplicative scale contributed by the prefixes, ignoring base-unit identity. For `si_none` / `iec_none` prefixes the factor is 1.0.

`bvn_unit_prefix_exponent` returns the sum of `(prefix_base_exponent × |unit_exponent|)` across all components.

## 12.4 SI Conversion API

Functions in `bovnar_si_units.h` provide dimensional analysis, compatibility checking, and value conversion between compatible physical units. **These functions reject currency units** — `bvn_find_si_conv` returns `NULL` for any `value_base_unit_t` for which `bvn_unit_is_currency` is true, causing `*ok = false`.

```

/* Full SI factor (physical units only) */
double bvn_unit_to_si_factor(value_unit_t u,
                             bool *is_affine,
                             double *affine_offset,
                             bool *ok);

/* SI dimension vector */
bool bvn_unit_dimension_vector(value_unit_t u,
                               int32_t dims[bvn_si_dim_count]);

/* Dimensional compatibility check */
bool bvn_units_compatible(value_unit_t a, value_unit_t b);

/* Conversion factor: value_in_b = value_in_a × k */
double bvn_unit_convert_factor(value_unit_t a, value_unit_t b,
                               bool *ok, bool *requires_affine);

/* Unit reduction */
value_unit_t bvn_unit_reduce(value_unit_t u,
                             double *scale, bool *overflow);

/* Prefix validity */
bool bvn_prefix_unit_valid(value_unit_prefix_t prefix,
                          value_base_unit_t base);

/* Exponent integer conversion */
int32_t bvn_exponent_to_int (unit_exponent_t e);
unit_exponent_t bvn_int_to_exponent(int32_t n);

```

For affine units (`bu_celsius`, `bu_fahrenheit`), `*is_affine` is set to `true` and `*affine_offset` receives the additive offset applied after multiplying by the returned factor. An affine unit is valid at exponent 1 only.

```

bool ok, affine;
double offset;
value_unit_t u = bvn_parse_unit((const uint8_t *) "°C", &ok);
double f = bvn_unit_to_si_factor(u, &affine, &offset, &ok);
/* f == 1.0, affine == true, offset == 273.15 */

```

## 12.5 Currency API

```

#include "bovnar_currency.h"

/* Classification */
bool bvn_unit_is_currency(int base);
bool bvn_unit_is_fiat    (int base);
bool bvn_unit_is_crypto  (int base);

/* Minor-unit exponent: 1 major unit = 10^N minor units */
uint8_t bvn_currency_minor_unit(int base, bool *ok);

/* Full currency metadata */
const bvn_currency_info_t *bvn_currency_info(int base);

/* Look up by 3–4 char code string; returns 0 (bu_none) on failure */
int bvn_parse_currency_str(const uint8_t *s, uint32_t len);

/* Prefix validity for a specific currency */
bool bvn_currency_prefix_valid(int base, int prefix_system);

```



The `bvn_currency_info_t` structure:

```
typedef struct {
    char      code[5];           /* "USD", "BTC", etc. */
    uint16_t  numeric_code;      /* ISO 4217 numeric code (0 for crypto) */
    uint8_t   minor_unit;        /* decimal places */
    bool      is_crypto;         /* true for cryptocurrencies */
    char      name[48];          /* "US Dollar", "Bitcoin", etc. */
} bvn_currency_info_t;
```

Examples:

```
bool ok;
uint8_t n = bvn_currency_minor_unit(bu_kwd, &ok); /* n=3, ok=true */
uint8_t m = bvn_currency_minor_unit(bu_jpy, &ok); /* m=0, ok=true */
uint8_t x = bvn_currency_minor_unit(bu_meter, &ok); /* x=0, ok=false */

const bvn_currency_info_t *ci = bvn_currency_info(bu_usd);
/* ci->code="USD", ci->numeric_code=840, ci->minor_unit=2,
   ci->is_crypto=false, ci->name="US Dollar" */

int cv = bvn_parse_currency_str((const uint8_t *)"EUR", 3); /* cv=176 */
int cc = bvn_parse_currency_str((const uint8_t *)"DOGE", 4); /* cc=323 */
int cx = bvn_parse_currency_str((const uint8_t *)"xyz", 3); /* cx=0 */

/* Distinguishing cup (volume) from CUP (currency) in code: */
value_unit_t volume = bvn_parse_unit((const uint8_t *)"cup", &ok);
value_unit_t currency = bvn_parse_unit((const uint8_t *)"CUP", &ok);
assert(!bvn_unit_is_currency(volume.components[0].base)); /* true */
assert( bvn_unit_is_currency(currency.components[0].base)); /* true */
```

## 12.6 Python API

```

from bovnar.enums import BaseUnit
from bovnar.currency import (
    is_currency, is_fiat, is_crypto,
    minor_unit, currency_info, currency_name, from_code,
    all_fiat, all_crypto,
)

assert is_currency(BaseUnit.USD)           # True
assert is_fiat(BaseUnit.XAU)               # True (gold is ISO 4217 X-code)
assert is_crypto(BaseUnit.ETH)             # True
assert not is_fiat(BaseUnit.BTC)           # True (BTC is crypto, not fiat)

assert minor_unit(BaseUnit.USD) == 2      # cents
assert minor_unit(BaseUnit.JPY) == 0      # indivisible
assert minor_unit(BaseUnit.KWD) == 3      # fils
assert minor_unit(BaseUnit.BTC) == 8      # satoshis
assert minor_unit(BaseUnit.ETH) == 18     # wei

info = currency_info(BaseUnit.EUR)
assert info.code == "EUR"
assert info.numeric_code == 978
assert info.minor_unit == 2

btc = from_code("BTC")
assert btc == BaseUnit.BTC

fiat_count = sum(1 for _ in all_fiat())    # 164
crypto_count = sum(1 for _ in all_crypto()) # 50

```

## 13. Integration with the Parser Event Stream

Unit information flows into the application through two paths:

1. **Type-annotation unit** — parsed from `<family:...,unit-param>` by the lexer, validated by the validator, delivered in the `ev_type_annotation_type_family_parameter` unit event.
2. **Inline unit suffix** — parsed from the suffix following a scalar value literal before the terminating `;`.

In both cases the effective unit is reported in the `bovnr_data_t.value_unit` field of the `ev_data` event.

## Full event sequence — physical unit

```
Input: .force = <float:64,k~g·m/s²> 9.81;

ev_assignment_start      data = "force"
ev_type_annotation_start data = "float:64,k~g·m/s²"
ev_type_annotation_type_family data = "float"
ev_type_annotation_type_family_parameter ← width=64
ev_type_annotation_type_family_parameter ← unit:
    value_unit = { num_components=3,
        [0] bu_gram,  exp_linear,  {prefix_si, si_kilo}
        [1] bu_meter, exp_linear,  {prefix_si, si_none}
        [2] bu_second, exp_neg_square, {prefix_si, si_none} }
ev_type_annotation_end
ev_data data="9.81"
```

## Full event sequence — currency unit

```
Input: .price = <float_dec:64,$USD> 19.99;

ev_assignment_start      data = "price"
ev_type_annotation_start data = "float_dec:64,USD"
ev_type_annotation_type_family data = "float_dec"
ev_type_annotation_type_family_parameter ← width=64
ev_type_annotation_type_family_parameter ← unit:
    value_unit = { num_components=1,
        [0] bu_usd, exp_linear, {prefix_si, si_none} }
ev_type_annotation_end
ev_data data="19.99"
```

For events with an explicit type annotation but no unit parameter, the unit event is **not emitted**. For `no_unit`, the unit event IS emitted with `BVN_UNIT_NONE (num_components=0)`. For synthesised (default) type annotations, the unit event IS emitted with `BVN_UNIT_NO_PREFIX(bu_none)`.

## Inline unit suffix — event stream view

```
Input: .distance = 1500 m;

ev_assignment_start      data = "distance"
ev_type_annotation_start data = "uint"      ← synthesised
ev_type_annotation_type_family data = "uint"
ev_type_annotation_type_family_parameter ← width=64 (synthesised)
ev_type_annotation_type_family_parameter ← base=10 (synthesised)
ev_type_annotation_type_family_parameter ← unit: BVN_UNIT_NO_PREFIX(bu_none)
ev_type_annotation_end
ev_data data="1500"
    value_unit = { num_components=1,
        [0] bu_meter, exp_linear, {prefix_si, si_none} }
```

The `value_unit` field of `ev_data` always reflects the final, reconciled unit.

## Practical callback

```
bool my_verified_handler(void* userdata, bvnr_event_t ev, bvnr_data_t* d)
{
    if (ev != ev_type_annotation_type_family_parameter)
        return true;

    value_unit_t u = d->value_unit;
    if (u.num_components == 0 ||
        (u.num_components == 1 && u.components[0].base == bu_none))
        return true;

    if (bvn_unit_is_currency(u.components[0].base)) {
        const bvn_currency_info_t *ci =
            bvn_currency_info(u.components[0].base);
        printf("currency: %s minor_unit=%u\n",
            ci->code, ci->minor_unit);
        return true;
    }

    char unit_str[128];
    bvn_unit_to_string(u, unit_str, sizeof(unit_str));
    printf("unit: %s (prefix_factor: %g)\n",
        unit_str, bvn_unit_prefix_factor(u));
    return true;
}
```

## 14. Validation Errors

The validator raises the following unit-specific errors:

Error code	Value	Trigger condition
<code>error_unit_illegal</code>	32	Unparseable unit string: unknown prefix, unknown base unit, unknown currency code after <code>\$</code> , a bare token in neither the physical-unit table nor (lacking the <code>\$</code> sigil) recognised as a currency (e.g. <code>XYZ</code> ), or bare <code>USD</code> ), invalid prefix-unit combination (e.g. IEC prefix on a currency, sub-kilo SI prefix on byte), empty component between separators (e.g. <code>m//s</code> ), or more than 8 components
<code>error_unit_too_long</code>	22	Unit string exceeds the internal type-buffer size limit
<code>error_unit_mismatch</code>	38	An inline unit suffix and an explicit type-annotation unit are both present, but parse to different <code>value_unit_t</code> representations
<code>error_unexpected_input_byte</code>	15	An inline unit suffix appears inside an array element

All four errors are raised during the `on_unverified` → validator phase. In `continue_on_error` mode the parser invokes `on_error` and enters the resync state machine, which skips to the next `;` at the current nesting depth. `bvnr_reader_get_error`, `..._line`, `..._column`, `..._byte`, and `..._offset` all report the location of the offending token.

## 15. Annotated Examples

### 15.1 Physical Quantities

```
# Thermodynamic temperature
.ambient_temp = <float:64,K>          293.15;

# Temperature in Celsius (affine: K = °C + 273.15)
.room_temp    = <float:32,°C>         20.0;

# Velocity and acceleration
.wind_speed   = <float:64,m/s>        12.5;
.gravity      = <float:64,m/s²>       9.80665;
.gravity_asc   = <float:64,m/s²>       9.80665; # ASCII caret – identical

# Pressure and energy
.tire_pressure = <float:32,k-Pa>       250.0;
.heat_energy   = <float:64,k~J>       5400.0;

# Flow rate (liters per minute)
.pump_flow     = <float:32,L/min>      15.0;

# Angles
.bearing       = <float:64,°>         270.0;
.phase         = <float:64,rad>       1.5708;

# Volume – US culinary
.recipe_water  = <float_dec:32,cup>    2.0; # "cup" (lowercase) = US cup
.recipe_flour  = <float_dec:32,tbsp>    3.0;

# Duration
.shelf_life    = <uint:32,wk>          52;
.service_life  = <float:64,yr>        10.0;
```

### 15.2 Digital Storage

```
.packet_size = <uint:32,B>          1500;
.cache_size  = <uint:64,Ki~B>       512;
.ram_size     = <uint:64,Mi~B>      4096;
.disk_size    = <uint:64,Gi~B>       500;
.link_rate    = <uint:32,M~b>       1000;
.nic_speed    = <float:64,G~b/s>    10.0;
```

### 15.3 Compound SI Quantities

```
# Force: Newton = k~g·m/s²
.force        = <float:64,k~g·m/s²>   9.81;
.force_alt    = <float:64,k~g·m·s⁻²>  9.81; # identical internal form

# Energy: Joule = k~g·m²/s²
.kinetic_energy = <float:64,k~g·m²/s²> 1000.0;

# Electric field
.field_strength = <float:64,V/m>       150.0;

# Torque
.torque        = <float:64,N·m>        25.0;
```

## 15.4 Currency Amounts and Rates

```
# — Fiat scalar amounts —————
.price_usd      = <float_dec:64,$USD>    19.99;
.balance_eur     = <float_dec:64,$EUR>   342.00;
.yen_fee        = <uint:64,$JPY>         500;    # zero minor unit – integer only
.kwd_invoice     = <uint:64,$KWD>        3500;    # 3.500 KWD in fils

# "CUP" (uppercase) is the Cuban Peso, NOT the US cup volume unit:
.cup_balance     = <float_dec:64,$CUP>    25.00;    # 25 Cuban Pesos

# — Crypto scalar amounts —————
.btc_sat        = <uint:64,$BTC>    54782000;    # on-chain satoshis
.eth_readable    = <float_dec:64,$ETH>    2.5;
.doge_bag        = <float_dec:64,$DOGE> 42000.0;
.usdt_stable     = <float_dec:64,$USDT> 5000.00;

# — Compound units —————
.gold_price      = <float_dec:64,$USD/oz_t> 2351.40; # $/troy oz
.wheat          = <float_dec:64,$USD/bsh>  5.82; # $/bushel
.rent           = <float_dec:64,$EUR/m²>   12.50; # €/m²
.billing_rate    = <float_dec:64,$EUR/h>   150.00; # €/h
.eur_usd         = <float_dec:64,$USD/$EUR> 1.0842; # exchange rate

# — Reporting scale —————
.fund_nav        = <float_dec:64,k~$USD>   250.0;    # $250,000
.gdp             = <float_dec:64,M~$EUR> 42800.0;    # €42.8 billion

# — Exchange rate with timestamp —————
.snapshot = {
    .epoch      = <uint:64,s>          1716400000;
    .eur_usd     = <float_dec:64,$USD/$EUR> 1.0842;
};

# — Array of prices —————
.tier_prices     = <float_dec:64,$USD> [9.99, 19.99, 49.99, 99.99];
```

## 15.5 Error Cases

```
# Empty component → error_unit_illegal
.bad1 = <float:64,m//s>      1.0;

# Too many components (9 > 8) → error_unit_illegal
.bad2 = <float:64,m*s*k~g*A*K*mol*cd*b*B> 1.0;

# Unknown base unit → error_unit_illegal
.bad3 = <float:64,foobar>    1.0;

# IEC prefix on a currency → error_unit_illegal
.bad4 = <float_dec:64,Ki~$USD> 1.0;

# Annotation unit differs from inline unit → error_unit_mismatch
.bad5 = <float:64,m> 1.0 s;

# Inline unit inside an array → error_unexpected_input_byte
.bad6 = <float:64,m> [1.0 m, 2.0 m];    # ERROR: suffix inside array

# Correct: dimensionless explicit
.ok1  = <uint:32,no_unit>    42;

# Correct: omitted unit (same behaviour as no_unit)
.ok2  = <uint:32>            42;

# Correct: BTU is a valid alias for bu_btu (currency lookup returns 0, physical table
# matches)
.ok3  = <float:64,BTU>      1.0;    # valid: same as Btu or btu

# Correct: sub-kilo SI prefix on currency is accepted
.ok4  = <float_dec:64,m~$USD> 0.001; # valid: milli-dollar (one tenth of a cent)

# Correct: cup (volume) vs CUP (currency) – both valid, different meaning
.vol  = <float_dec:32,cup>   2.0;    # US cup (236.6 mL)
.bal  = <float_dec:64,$CUP> 25.00;   # Cuban Peso
```

---

*End of Bovnar Quantity Annotation System — Unit and Currency Reference, v1.0.*