

# Metro

fenjalien and Mc-Zen

<https://github.com/fenjalien/typst-units>

Version 0.1.0

## Contents

1 Introduction .....	2
2 Usage .....	2
2.1 Options .....	2
2.2 Numbers .....	2
2.2.1 Options .....	2
2.3 Units .....	5
2.3.1 Options .....	7
2.4 Quantities .....	8
2.4.1 Options .....	9
3 Meet the Units .....	9
4 Creating .....	13
4.1 Units .....	13
4.2 Prefixes .....	13
4.3 Powers .....	13
4.4 Qualifiers .....	14

# 1 Introduction

The Metro package aims to be a port of the Latex package siunitx. It allows easy typesetting of numbers and units with options. This package is very early in development and many features are missing, so any feature requests or bug reports are welcome!

Metro's name comes from Metrology, the study scientific study of measurement.

## 2 Usage

### 2.1 Options

```
#metro-setup(..options)
```

Options for Metro's can be modified by using the `metro-setup` function. It takes an argument `sink` and saves any named parameters found. The options for each function are specified in their respective sections.

All options and function parameters use the following types:

**Literal** Takes the given value directly. Input type is a string, content and sometimes a number.

**Switch** On-off switches. Input type is a boolean.

**Choice** Takes a limited number of choices, which are described separately for each option. Input type is a string.

**Number** Takes a float or integer.

### 2.2 Numbers

```
#num(number, e: none, pm: none, ..options)
```

Formats a number. Exponents and uncertainties can be given using the named parameters. All parameters listed can be given as a string, content or a number. Note that number parsing is very limited and fragile at the moment.

Also note that explicitly written parts of a number when using a number type will be lost as Typst automatically parses them.

**number** **Literal**

The number to format.

**pm** **Literal**

Uncertainty.

(default: none)

**e** **Literal**

Exponent.

(default: none)

123	<code>#num(123)</code>
1234	<code>#num("1234")</code>
12 345	<code>#num[12345]</code>
0.123	<code>#num(0.123)</code>
0.1234	<code>#num("0,1234")</code>
0.123 45	<code>#num[.12345]</code>
$3.45 \times 10^{-4}$	<code>#num(e: -4)[3.45]</code>
$-10^{10}$	<code>#num("-1", e: 10, print-unity-mantissa: false)</code>

#### 2.2.1 Options

**input-decimal-markers** `Array<Literal>`

(default: `('\.', ',')`)

An array of characters that indicate the separation between the integer and decimal parts of a number. More than one input decimal marker can be used, it will be converted by the package to the appropriate output marker.

**retain-explicit-decimal-marker** Switch (default: false)

Allows a trailing decimal marker with no decimal part present to be printed.

```
10          #num[10.]
10.         #num(retain-explicit-decimal-marker: true)[10.]
```

**retain-explicit-plus** Switch (default: false)

Allows a leading plus sign to be printed.

```
345         #num[+345]
+345        #num(retain-explicit-plus: true)[+345]
```

**retain-negative-zero** Switch (default: false)

Allows a negative sign on an entirely zero value.

```
0          #num[-0]
-0         #num(retain-negative-zero: true)[-0]
```

**minimum-decimal-digits** Integer (default: 0)

May be used to pad the decimal component of a number to a given size.

```
0.123      #num(0.123)
0.123      #num(0.123, minimum-decimal-digits: 2)
0.1230     #num(0.123, minimum-decimal-digits: 4)
```

**minimum-integer-digits** Integer (default: 0)

May be used to pad the integer component of a number to a given size.

```
123        #num(123)
123        #num(123, minimum-integer-digits: 2)
0123       #num(123, minimum-integer-digits: 4)
```

**group-digits** Choice (default: "all")

Whether to group digits into blocks to increase the ease of reading of numbers. Takes the values all, none, decimal and integer. Grouping can be activated separately for the integer and decimal parts of a number using the appropriately named values.

```
12 345.678 90    #num[12345.67890]
12345.67890      #num(group-digits: "none")[12345.67890]
12345.678 90     #num(group-digits: "decimal")[12345.67890]
12 345.67890     #num(group-digits: "integer")[12345.67890]
```

**group-separator** Literal (default: sym.space.thin)

The separator to use between groups of digits.

```
12 345          #num[12345]
12,345          #num(group-separator: ",")[12345]
12 345          #num(group-separator: " ")[12345]
```

**group-minimum-digits** Integer (default: 5)

Controls how many digits must be present before grouping is applied. The number of digits is considered separately for the integer and decimal parts of the number: grouping does not “cross the boundary”.

1234	<code>#num[1234]</code>
12 345	<code>#num[12345]</code>
1 234	<code>#num(group-minimum-digits: 4)[1234]</code>
12 345	<code>#num(group-minimum-digits: 4)[12345]</code>
1234.5678	<code>#num[1234.5678]</code>
12 345.678 90	<code>#num[12345.67890]</code>
1 234.567 8	<code>#num(group-minimum-digits: 4)[1234.5678]</code>
12 345.678 90	<code>#num(group-minimum-digits: 4)[12345.67890]</code>

**digit-group-size** Integer (default: 3)

Controls the number of digits in each group. Finer control can be achieved using `digit-group-first-size` and `digit-group-other-size`: the first group is that immediately by the decimal point, the other value applies to the second and subsequent groupings.

1 234 567 890	<code>#num[1234567890]</code>
12345 67890	<code>#num(digit-group-size: 5)[1234567890]</code>
1 23 45 67 890	<code>#num(digit-group-other-size: 2)[1234567890]</code>

**output-decimal-marker** Literal (default: .)

The decimal marker used in the output. This can differ from the input marker.

1.23	<code>#num(1.23)</code>
1,23	<code>#num(output-decimal-marker: ",")[1.23]</code>

**exponent-base** Literal (default: 10)

The base of an exponent.

$1 \times 2^2$	<code>#num(exponent-base: "2", e: 2)[1]</code>
----------------	--

**exponent-product** Literal (default: `sym.times`)

The symbol to use as the product between the number and its exponent.

$1 \times 10^2$	<code>#num(e: 2, exponent-product: sym.times)[1]</code>
$1 \cdot 10^2$	<code>#num(e: 2, exponent-product: sym.dot)[1]</code>

**output-exponent-marker** Literal (default: none)

When not none, the value stored will be used in place of the normal product and base combination.

1e2	<code>#num(output-exponent-marker: "e", e: 2)[1]</code>
1E2	<code>#num(output-exponent-marker: "E", e: 2)[1]</code>

**bracket-ambiguous-numbers** Switch (default: true)

There are certain combinations of numerical input which can be ambiguous. This can be corrected by adding brackets in the appropriate place.

$(1.2 \pm 0.3) \times 10^4$	<code>#num(e: 4, pm: 0.3)[1.2]</code>
$1.2 \pm 0.3 \times 10^4$	<code>#num(bracket-ambiguous-numbers: false, e: 4, pm: 0.3)[1.2]</code>

**bracket-negative-numbers** Switch (default: false)

Whether or not to display negative numbers in brackets.

-15 673	<code>#num[-15673]</code>
(15 673)	<code>#num(bracket-negative-numbers: true)[-15673]</code>

**tight-spacing** Switch (default: false)

Compresses spacing where possible.

$2 \times 10^3$	<code>#num(e: 3)[2]</code>
$2 \times 10^3$	<code>#num(e: 3, tight-spacing: true)[2]</code>

### **print-implicit-plus** Switch (default: false)

Force the number to have a sign. This is used if given and if no sign was present in the input.

345	<code>#num(345)</code>
+345	<code>#num(345, print-implicit-plus: true)</code>

It is possible to set this behaviour for the exponent and mantissa independently using `print-mantissa-implicit-plus` and `print-exponent-implicit-plus` respectively.

### **print-unity-mantissa** Switch (default: true)

Controls the printing of a mantissa of 1.

$1 \times 10^4$	<code>#num(e: 4)[1]</code>
$10^4$	<code>#num(e: 4, print-unity-mantissa: false)[1]</code>

### **print-zero-exponent** Switch (default: false)

Controls the printing of an exponent of 0.

444	<code>#num(e: 0)[444]</code>
$444 \times 10^0$	<code>#num(e: 0, print-zero-exponent: true)[444]</code>

### **print-zero-integer** Switch (default: true)

Controls the printing of an integer component of 0.

0.123	<code>#num(0.123)</code>
.123	<code>#num(0.123, print-zero-integer: false)</code>

### **zero-decimal-as-symbol** Switch (default: false)

Whether to show entirely zero decimal parts as a symbol. Uses the symbol stroed using `zero-symbol` as the replacement.

123.00	<code>#num[123.00]</code>
123.—	<code>#metro-setup(zero-decimal-as-symbol: true)</code>
123.[—]	<code>#num[123.00]</code>
	<code>#num(zero-symbol: [#sym.bar.h])[123.00]</code>

### **zero-symbol** Literal (default: `sym.bar.h`)

The symbol to use when `zero-decimal-as-symbol` is true.

## 2.3 Units

`#unit(unit, ..options)`

Typsets a unit and provides full control over output format for the unit. The type passed to the function can be either a string or some math content.

When using math Typst accepts single characters but multiple characters together are expected to be variables. So Metro defines units and prefixes which you can import to be use.

```
#import "@preview/metro:0.1.0": unit, units, prefixes
#unit($units.kg m/s^2$)
// because `units` and `prefixes` here are modules you can import what you need
#import units: gram, metre, second
#import prefixes: kilo
$unit(kilo gram metre / second^2)$
// You can also just import everything instead
#import units: *
```

```
#import prefixes: *
$unit(joule / mole / kelvin)$
```

$$\text{kg m s}^{-2}$$

$$\text{kg m s}^{-2}$$

$$\text{J mol}^{-1} \text{ K}^{-1}$$

When using strings there is no need to import any units or prefixes as the string is parsed. Additionally several variables have been defined to allow the string to be more human readable. You can also use the same syntax as with math mode.

```
// String
#unit("kilo gram metre per square second")
// Math equivalent
#unit($kilo gram metre / second^2$)
// String using math syntax
#unit("kilo gram metre / second^2")
```

$$\text{kg m s}^{-2}$$

$$\text{kg m s}^{-2}$$

$$\text{kg m s}^{-2}$$

per used as in “metres *per* second” is equivalent to a slash /. When using this in a string you don’t need to specify a numerator.

```
#unit("metre per second")
$unit(metre/second)$
```

```
#unit("per square becquerel")
#unit("/becquerel^2")
```

$$\text{m s}^{-1}$$

$$\text{m s}^{-1}$$

$$\text{Bq}^{-2}$$

$$\text{Bq}^{-2}$$

square and cubic apply their respective powers to the units after them, while squared and cubed apply to units before them.

```
#unit("square becquerel")
#unit("joule squared per lumen")
#unit("cubic lux volt tesla cubed")
```

$$\text{Bq}^2$$

$$\text{J}^2 \text{ lm}^{-1}$$

$$\text{lx}^3 \text{ V T}^3$$

Generic powers can be inserted using the tothe and raiseto functions. tothe specifically is equivalent to using caret ^.

```
#unit("henry tothe(5)")
#unit($henry^5$)
#unit("henry^5")
```

```
#unit("raiseto(4.5) radian")
#unit($radian^4.5$)
#unit("radian^4.5")
```

$$\text{H}^5$$

$$\text{H}^5$$

$$\text{H}^5$$

rad<sup>4.5</sup>  
rad<sup>4.5</sup>  
rad<sup>4.5</sup>

Generic qualifiers are available using the `of` function which is equivalent to using an underscore `_`. Note that when using an underscore for qualifiers in a string with a space, to capture the whole qualifier use brackets `()`.

```
#unit("kilogram of(metal)")
#unit($kilogram_"metal"$)
#unit("kilogram_metal")

#metro-setup(qualifier-mode: "bracket")
#unit("milli mole of(cat) per kilogram of(prod)")
#unit($milli mole_"cat" / kilogram_"prod"$)
#unit("milli mole_(cat) / kilogram_(prod)")

kgmetal
kgmetal
kgmetal

mmol(cat) kg(prod)-1
mmol(cat) kg(prod)-1
mmol(cat) kg(prod)-1
```

### 2.3.1 Options

**inter-unit-product** Literal (default: `sym.space.thin`)

The separator between each unit. The default setting is a thin space: another common choice is a centred dot.

```
F2 lm cd      #unit("farad squared lumen candela")
F2 · lm · cd  #unit("farad squared lumen candela", inter-unit-product: $dot.c$)
```

**per-mode** Choice (default: `"power"`)

Use to alter the handling of `per`.

**power** Reciprocal powers

```
J mol-1 K-1      #unit("joule per mole per kelvin")
m s-2           #unit("metre per second squared")
```

**fraction** Uses the `math.frac` function (also known as `$ / $`) to typeset positive and negative powers of a unit separately.

```
 $\frac{\text{J}}{\text{mol K}}$       #unit("joule per mole per kelvin", per-mode: "fraction")
 $\frac{\text{m}}{\text{s}^2}$         #unit("metre per second squared", per-mode: "fraction")
```

**symbol** Separates the two parts of a unit using the symbol in `per-symbol`. This method for displaying units can be ambiguous, and so brackets are added unless `bracket-unit-denominator` is set to `false`. Notice that `bracket-unit-denominator` only applies when `per-mode` is set to `symbol`.

```
J/(mol K)      #metro-setup(per-mode: "symbol")
m/s2          #unit("joule per mole per kelvin")
               #unit("metre per second squared")
```

**per-symbol** Literal (default: `sym.slash`)

The symbol to use to separate the two parts of a unit when `per-symbol` is `"symbol"`.

```
#unit("joule per mole per kelvin", per-mode: "symbol", per-symbol: [ div ])
J div (mol K)
```

**bracket-unit-denominator** Switch (default: true)

Whether or not to add brackets to unit denominators when per-symbol is "symbol".

```
#unit("joule per mole per kelvin", per-mode: "symbol", bracket-unit-denominator:
false)
J/mol K
```

**sticky-per** Switch (default: false)

Normally, per applies only to the next unit given. When sticky-per is true, this behaviour is changed so that per applies to all subsequent units.

```
Pa Gy-1 H          #unit("pascal per gray henry")
Pa Gy-1 H-1        #unit("pascal per gray henry", sticky-per: true)
```

**qualifier-mode** Choice (default: "subscript")

Sets how unit qualifiers can be printed.

**subscript**

```
#unit("kilogram of(pol) squared per mole of(cat) per hour")
kg(pol)2 mol(cat)-1 h-1
```

**bracket**

```
#unit("kilogram of(pol) squared per mole of(cat) per hour", qualifier-mode:
"bracket")
kg(pol)2 mol(cat)-1 h-1
```

**combine** Powers can lead to ambiguity and are automatically detected and brackets added as appropriate.

```
dBi          #unit("deci bel of(i)", qualifier-mode: "combine")
```

**phrase** Used with qualifier-phrase, which allows for example a space or other linking text to be inserted.

```
#metro-setup(qualifier-mode: "phrase", qualifier-phrase: sym.space)
#unit("kilogram of(pol) squared per mole of(cat) per hour")
#metro-setup(qualifier-phrase: [ of ])
#unit("kilogram of(pol) squared per mole of(cat) per hour")
kg pol2 mol cat-1 h-1
kg of pol2 mol of cat-1 h-1
```

**power-half-as-sqrt** Switch (default: false)

When true the power of 0.5 is shown by giving the unit symbol as a square root.

```
Hz0.5          #unit("Hz tothe(0.5)")
√Hz            #unit("Hz tothe(0.5)", power-half-as-sqrt: true)
```

## 2.4 Quantities

```
#qty(number, unit, ..options)
```

This function combines the functionality of num and unit and formats the number and unit together. The number and unit arguments work exactly like those for the num and unit functions respectively.

```
1.23 J mol-1 K-1
0.23 × 107 cd
```



1.99 /kg	<code>#qty(1.23, "J / mol / kelvin")</code>
1.345 $\frac{\text{C}}{\text{mol}}$	<code>\$qty(.23, candela, e: 7)\$</code>
	<code>#qty(1.99, "per kilogram", per-mode: "symbol")</code>
	<code>#qty(1.345, "C/mol", per-mode: "fraction")</code>

### 2.4.1 Options

**allow-quantity-breaks** Switch (default: false)

Controls whether the combination of the number and unit can be split across lines.

```
#box(width: 4.5cm)[Some filler text #qty(10, "m")]
#metro-setup(allow-quantity-breaks: true)
#box(width: 4.5cm)[Some filler text #qty(10, "m")]
```

Some filler text

10 m

Some filler text 10

m

**quantity-product** Literal (default: `sym.space.thin`)

The product symbol between the number and unit.

```
#qty(2.67, "farad")
#qty(2.67, "farad", quantity-product: sym.space)
#qty(2.67, "farad", quantity-product: none)
```

2.67 F

2.67 F

2.67F

## 3 Meet the Units

The following tables show the currently supported prefixes, units and their abbreviations. Note that unit abbreviations that have single letter commands are not available for import for use in math.

This is because math mode already accepts single letter variables.

Unit	Command	Symbol
ampere	<code>ampere</code>	A
candela	<code>candela</code>	cd
kelvin	<code>kelvin</code>	K
kilogram	<code>kilogram</code>	kg
metre	<code>metre</code>	m
mole	<code>mole</code>	mol
second	<code>second</code>	s

Table 1: SI base units.

Unit	Command	Symbol	Unit	Command	Symbol
becquerel	becquerel	Bq	newton	newton	N
degree Celsius	degreeCelsius	°C	ohm	ohm	Ω
coulomb	coulomb	C	pascal	pascal	Pa
farad	farad	F	radian	radian	rad
gray	gray	Gy	siemens	siemens	S
hertz	hertz	Hz	sievert	sievert	Sv
henry	henry	H	steradian	steradian	sr
joule	joule	J	tesla	tesla	T
lumen	lumen	lm	volt	volt	V
katal	katal	kat	watt	watt	W
lux	lux	lx	weber	weber	Wb

Table 2: Coherent derived units in the SI with special names and symbols.

Unit	Command	Symbol
astronomicalunit	astronomicalunit	au
bel	bel	B
dalton	dalton	Da
day	day	d
decibel	decibel	dB
degree	degree	°
electronvolt	electronvolt	eV
hectare	hectare	ha
hour	hour	h
litre	litre	L
	liter	L
minute (plane angle)	arcminute	'
minute (time)	minute	min
second (plane angle)	arcsecond	"
neper	neper	Np
tonne	tonne	t

Table 3: Non-SI units accepted for use with the International System of Units.

Prefix	Command	Symbol	Power	Prefix	Command	Symbol	Power
quecto	quecto	q	−30	deca	deca	da	1
ronto	ronto	r	−27	hecto	hecto	h	2
yocto	yocto	y	−24	kilo	kilo	k	3
atto	atto	a	−21	mega	mega	M	6
zepto	zepto	z	−18	giga	giga	G	9
femto	femto	f	−15	tera	tera	T	12
pico	pico	p	−12	peta	peta	P	15
nano	nano	n	−9	exa	exa	E	18
micro	micro	μ	−6	zetta	zetta	Z	21
milli	milli	m	−3	yotta	yotta	Y	24
centi	centi	c	−2	ronna	ronna	R	27
deci	deci	d	−1	quetta	quetta	Q	30

Table 4: SI prefixes

Unit	Abbreviation	Symbol	Unit	Abbreviation	Symbol	Unit	Abbreviation	Symbol
femtogram	fg	fg	millihertz	mHz	mHz	farad	F	F
picogram	pg	pg	hertz	Hz	Hz	femtofarad	fF	fF
nanogram	ng	ng	kilohertz	kHz	kHz	picofarad	pF	pF
microgram	ug	μg	megahertz	MHz	MHz	nanofarad	nF	nF
milligram	mg	mg	gigahertz	GHz	GHz	microfarad	uF	μF
gram	g	g	terahertz	THz	THz	millifarad	mF	mF
kilogram	kg	kg	millinewton	mN	mN	henry	H	H
picometre	pm	pm	newton	N	N	femtohenry	fH	fH
nanometre	nm	nm	kilonewton	kN	kN	picohenry	pH	pH
micrometre	um	μm	meganewton	MN	MN	nanohenry	nH	nH
millimetre	mm	mm	pascal	Pa	Pa	millihenry	mH	mH
centimetre	cm	cm	kilopascal	kPa	kPa	microhenry	uH	μH
decimetre	dm	dm	megapascal	MPa	MPa	coulomb	C	C
metre	m	m	gigapascal	GPa	GPa	nanocoulomb	nC	nC
kilometre	km	km	milliohm	mohm	mΩ	millicoulomb	mC	mC
attosecond	as	as	kilohm	kohm	kΩ	microcoulomb	uC	μC
femtosecond	fs	fs	megohm	Mohm	MΩ	kelvin	K	K
picosecond	ps	ps	picovolt	pV	pV	decibel	dB	dB
nanosecond	ns	ns	nanovolt	nV	nV	astronomicalunit	au	au
microsecond	us	μs	microvolt	uV	μV	becquerel	Bq	Bq
millisecond	ms	ms	millivolt	mV	mV	candela	cd	cd
second	s	s	volt	V	V	dalton	Da	Da
femtomole	fmol	fmol	kilovolt	kV	kV	gray	Gy	Gy
picomole	pmol	pmol	watt	W	W	hectare	ha	ha
nanomole	nmol	nmol	nanowatt	nW	nW	katal	kat	kat
micromole	umol	μmol	microwatt	uW	μW	lumen	lm	lm
millimole	mmol	mmol	milliwatt	mW	mW	neper	Np	Np
mole	mol	mol	kilowatt	kW	kW	radian	rad	rad
kilomole	kmol	kmol	megawatt	MW	MW	sievert	Sv	Sv
picoampere	pA	pA	gigawatt	GW	GW	steradian	sr	sr
nanoampere	nA	nA	joule	J	J	weber	Wb	Wb
microampere	uA	μA	microjoule	uJ	uJ			
milliampere	mA	mA	millijoule	mJ	mJ			
ampere	A	A	kilojoule	kJ	kJ			
kiloampere	kA	kA	electronvolt	eV	eV			
microlitre	uL	μL	millielectronvolt	meV	meV			
millilitre	mL	mL	kiloelectronvolt	keV	keV			
litre	L	L	megaelectronvolt	MeV	MeV			
hectolitre	hL	hL	gigaelectronvolt	GeV	GeV			
			teraelectronvolt	TeV	TeV			
			kilowatt hour	kWh	kWh			

Table 5: Unit abbreviations

## 4 Creating

The following functions can be used to define custom units, prefixes, powers and qualifiers that can be used with the unit function.

### 4.1 Units

`#declare-unit(unit, symbol, ..options)`

Declare's a custom unit to be used with the unit and qty functions.

**unit** string

The string to use to identify the unit for string input.

**symbol** Literal

The unit's symbol. A string or math content can be used. When using math content it is recommended to pass it through unit first.

```
#let inch = "in"
#declare-unit("inch", inch)
#unit("inch / s")
#unit($inch / s$)
in s-1
in s-1
```

### 4.2 Prefixes

`#create-prefix(symbol)`

Use this function to correctly create the symbol for a prefix. Metro uses Typst's `math.class` function with the class parameter "unary" to designate a prefix. This function does it for you.

**symbol** Literal

The prefix's symbol. A string or math content can be used. When using math content it is recommended to pass it through unit first.

`#declare-prefix(prefix, symbol, power-tens)`

Declare's a custom prefix to be used with the unit and qty functions.

**prefix** string

The string to use to identify the prefix for string input.

**symbol** Literal

The prefix's symbol. This should be the output of the create-prefix function specified above.

**power-tens** Number

The power ten of the prefix.

```
#let myria = create-prefix("my")
#declare-prefix("myria", myria, 4)
#unit("myria meter")
#unit($myria meter$)
mym
mym
```

### 4.3 Powers

`#declare-power(before, after, power)`

This function adds two symbols for string input, one for use before a unit, the second for use after a unit, both of which are equivalent to the power.

**before** string

The string that specifies this power before a unit.

**after** string

The string that specifies this power after a unit.

**power** Number

The power.

```
#declare-power("quartic", "tothefourth", 4)
#unit("kilogram tothefourth")
#unit("quartic metre")
kg4
m4
```

## 4.4 Qualifiers

`#declare-qualifier(qualifier, symbol)`

This function defines a custom qualifier for string input.

**qualifier** string

The string that specifies this qualifier.

**symbol** Literal

The qualifier's symbol. Can be string or content.

```
#declare-qualifier("polymer", "pol")
#declare-qualifier("catalyst", "cat")
#unit("gram polymer per mole catalyst per hour")
gpol molcat-1 h-1
```