Metro

Contents

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.

number `Number`
The number to format.

pm `Literal` (default: `none`)
Uncertainty of the number.

e `Number` (default: `none`)
Exponent. The exponent is applied to both the number and the uncertainty if given.

123
−1234
12345

2.2.1 Options

times `c`
The symbol

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: *
$unit(joule / mole / kelvin)$
```

$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{J\,mol^{-1}\,K^{-1}}$

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")
```

$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{kg\,m\,s^{-2}}$

`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")
```

$\mathrm{m\,s^{-1}}$
$\mathrm{m\,s^{-1}}$

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

`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")
```

$\mathrm{Bq^{2}}$
$\mathrm{J^{2}\,lm^{-1}}$
$\mathrm{lx^{3}\,V\,T^{3}}$

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")
```

H^{5}
H^{5}
H^{5}

$\mathrm{rad}^{4.5}$
$\mathrm{rad}^{4.5}$
$\mathrm{rad}^{4.5}$

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)")
```

$\mathrm{kg}_{\mathrm{metal}}$
$\mathrm{kg}_{\mathrm{metal}}$
$\mathrm{kg}_{\mathrm{metal}}$

$\mathrm{mmol(cat)\,kg(prod)^{-1}}$
$\mathrm{mmol(cat)\,kg(prod)^{-1}}$
$\mathrm{mmol(cat)\,kg(prod)^{-1}}$

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.

```
#unit("farad squared lumen candela")\
#unit("farad squared lumen candela", inter-unit-product: $dot.c$)
```

$\mathrm{F}^{2}\ \mathrm{lm}\ \mathrm{cd}$
$\mathrm{F}^{2} \cdot \mathrm{lm} \cdot \mathrm{cd}$

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

Use to alter the handling of `per`.

power Reciprocal powers

```
#unit("joule per mole per kelvin")\
#unit("metre per second squared")
```

$\mathrm{J\,mol^{-1}\,K^{-1}}$
$\mathrm{m\,s^{-2}}$

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

```
#unit("joule per mole per kelvin", per-mode: "fraction")\
#unit("metre per second squared", per-mode: "fraction")
```

$$\frac{\mathrm{J}}{\mathrm{mol\,K}}$$
$$\frac{\mathrm{m}}{\mathrm{s^2}}$$

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.

```
#metro-setup(per-mode: "symbol")
#unit("joule per mole per kelvin")\
#unit("metre per second squared")
```

$\mathrm{J/(mol\,K)}$
$\mathrm{m/s^2}$

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 ])
```

$\mathrm{J\ div\ (mol\,K)}$

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)
```

$\mathrm{J/mol\,K}$

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.

```
#unit("pascal per gray henry")\
#unit("pascal per gray henry", sticky-per: true)
```

$\mathrm{Pa\,Gy^{-1}\,H}$
$\mathrm{Pa\,Gy^{-1}\,H^{-1}}$

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

Sets how unit qualifiers can be printed.

subscript

```
#unit("kilogram of(pol) squared per mole of(cat) per hour")
```

$\mathrm{kg^2_{pol}\,mol^{-1}_{cat}\,h^{-1}}$

bracket

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

$\mathrm{kg(pol)^2\,mol(cat)^{-1}\,h^{-1}}$

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

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

dBi

phrase Used with `qualifier-phrase`, which allows for example a space or othre 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")
```

$\mathrm{kg\ pol^{2}\ mol\ cat^{-1}\ h^{-1}}$
$\mathrm{kg\ of\ pol^{2}\ mol\ of\ cat^{-1}\ h^{-1}}$

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

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

```
#unit("Hz tothe(0.5)")\
#unit("Hz tothe(0.5)", power-half-as-sqrt: true)
```

$\mathrm{Hz^{0.5}}$
$\sqrt{\mathrm{Hz}}$

2.4 Quantities

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 it accepts single letters.

4 Creating

The following functions can be used to define cutom 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($inch / s$)
```

$\mathrm{in\ s^{-1}}$
$\mathrm{in\ s^{-1}}$

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 ues after a unit, obth 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")
```

kg^{4}
m^{4}

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")
```

$\mathrm{g_{pol}}\,\mathrm{mol_{cat}^{-1}}\,\mathrm{h}^{-1}$