Just SI Units

Release 0.2a2

David Scheliga

Jul 15, 2021

CONTENTS

justunits handles (SI-) units within string occurrences. The module's *justunits* tasks are limited to the interchangeability of different unit formatting styles. This module focus **just** on **units**, it does not provide unit conversion or anything related to such a topic. It's purpose is to provide an entry point for a future unit conversion within another module.

justunits major purpose can be shown with 2 graphs. It's first purpose is to provide a support for different value/attribute and unit separation styles, which can occur within text files.

Note: The current development state of this project is `alpha`. It's published for a first stress test of 2 major functions.

Towards the beta

- naming of modules, classes and methods will change, since the final wording is not done.
- Code inspections are not finished.
- The documentation is broad or incomplete.
- Testing is not complete, as it is added during the first test phase. At this state mostly doctests are applied at class or function level.

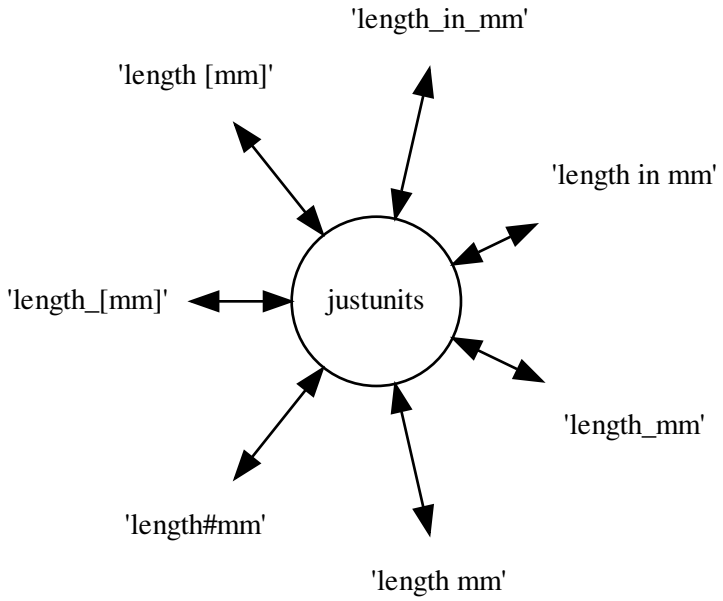

Figure 1: Supported separations between an attribute/value and a unit.

The second purpose is to detect SI-units and support 4 major layouts emphasising utf8 with superscripts using a dot as a separator.

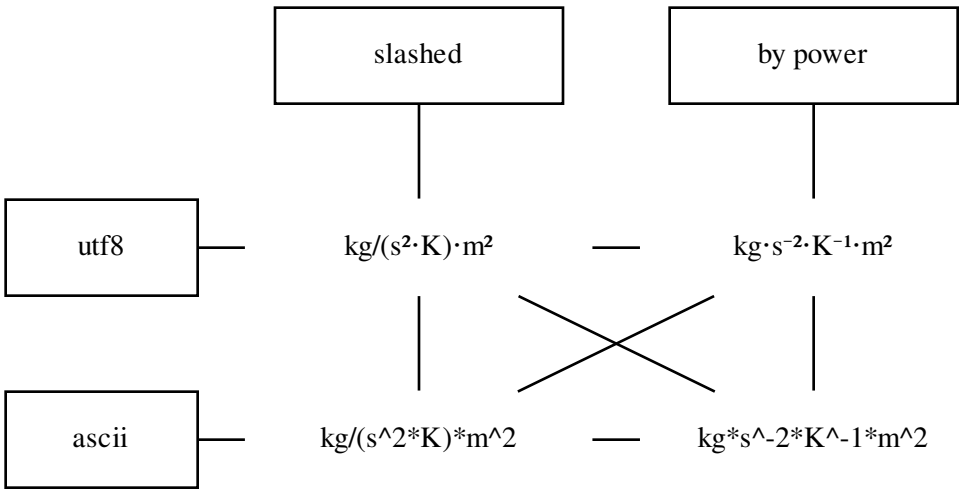

Figure 2: Unit representations supported by justunits.

CHAPTER

ONE

INSTALLATION

Install the latest release from pip.

```
$ pip install justunits
```

1.1 Basic Usage

1.1.1 Styling

You might want to set your desired attribute unit and unit style flavor first or just keep the preset defaults. See all supported unit styles.

```
>>> import justunits
>>> from justunits import AttributeUnitSeparators, UnitStyle
>>> justunits.set_default_attribute_unit_separator(AttributeUnitSeparators.WHITESPACE_
↪BOXED)
>>> justunits.set_default_unit_style(UnitStyle.FINE_SLASHED_SUPERSCRIPT)
```

1.1.2 Reformatting

Attribute/value-unit pairs come in many different, justifiable forms. Either they are demanded by preexisting file format definitions or depend on the context of information transfer. Transferring different formats depicted in figure 1.

```
>>> import justunits
>>> from justunits import AttributeUnitSeparators, UnitStyle, UnitDividingStyle
>>> justunits.reformat_unit("pressure#kg*m^-1*s^-2")
'pressure [kg/(ms²)]'
```

Change just the separator in between attribute and unit.

```
>>> justunits.reformat_unit(
...     "pressure#kg*m^-1*s^-2",
...     target_format=AttributeUnitSeparators.WHITESPACE_IN
... )
'pressure in kg/(ms²)'
```

Or make custom combinations. Missing definitions will be replaced by defaults.

```
>>> boxed_fine_powered = (
...     AttributeUnitSeparators.WHITESPACE_BOXED | UnitDividingStyle.BY_POWER
... )
>>> justunits.reformat_unit(
...     "pressure#kg*m^-1*s^-2",
...     target_format=boxed_fine_powered
... )
'pressure [kgm¹s²]'
```

1.1.3 Splitting

You may come across one of these exemplary formats and want to split attributes/values from units from further processing.

Header of a csv file:

```
time[s],energy[J],torque[Nm]
0.0,0.0,0.0
...
```

Attributes with units in text files:

```
...
height_mm = 1.0
width_mm = 2.0
...
```

Or values with units in text files:

```
...
height = 1.0 mm
width = 2.0 mm
...
```

split_unit_text

Splitting attributes/values from units preserving text. The function *justunits.split_unit_text()* provides a splitting whenever a unit detection is not desired.

```
>>> justunits.split_unit_text("heat capacity[kg/(s^2*K^-1)*m^2]")
('heat capacity', 'kg/(s^2*K^-1)*m^2')

>>> justunits.split_unit_text("1.23 apples")
('1.23', 'apples')
```

split_unit

justunits.split_unit() provides the main feature of detecting units in the process of splitting.

```
>>> attribute_name, any_units = justunits.split_unit("pressure#kg*m^-1*s^-2")
>>> attribute_name
'pressure'

The unit text is converted into an iterable DerivedUnit-Object containing all detected
(and unknown units).

>>> any_units
(AUnit(kg 'kilogram' mass, 1e+03g), AUnit(m 'meter' length; power=-1), AUnit(s 'second'␣
↪time; power=-2))

>>> [str(unit) for unit in any_units]
['kg', 'm^-1.0', 's^-2.0']
```

split_unit converting attribute/value

It is also possible to convert the split values by converting them with a *Callable[[str], Any]*`.

```
>>> justunits.split_unit("1.23 apples", converter=float)
(1.23, (UnknownUnit('apples'),))
```

Be aware that using a *conversion* function any exception will not be caught.

```
>>> justunits.split_unit("1.23 apples", converter=int)
Traceback (most recent call last):
...
ValueError: invalid literal for int() with base 10: '1.23'
```

This is intentional as the purpose of the converters result depends on your choice. Either catch a related exception or take different solutions.

```
>>> justunits.split_unit("1.23 apples", converter=lambda x: int(float(x)))
(1, (UnknownUnit('apples'),))
```

1.1.4 Joining

`justunits.join()` The counterpart of splitting attributes/values-unit pairs joins

```
>>> pressure = justunits.from_string("kg/(m*s^2)")
>>> justunits.join_unit("pressure", pressure)
'pressure [kg/(ms²)]'

>>> from justunits import AttributeUnitSeparators
>>> justunits.join_unit(
...     "pressure", pressure, target_format=AttributeUnitSeparators.WHITESPACE_IN
... )
'pressure in kg/(ms²)'
```

1.2 Unit styles of justunits

The formatting of units is sectioned into the separation of attribute/value-unit pairs and the styling of the units themself.

1.2.1 Defaults

justunits formats by these default definitions.

justunits.**DEFAULT_ATTRIBUTE_UNIT_SEPARATOR = AttributeUnitSeparators.WHITESPACE_IN**
The style separating attributes/values from units.

justunits.**DEFAULT_ALLOWED_SEPARATORS = AttributeUnitSeparators.None**
By default only 5 of 7 supported attribute unit separator are taken into account as the absolute certain separation of an single underline or whitespace separated unit is ambiguous, without limiting the results to the content of the unit library. Such a behaviour may be implemented in future releases, if it turns out to be necessary.

justunits.**DEFAULT_POWER_STYLE = UnitPowerStyle.SUPERSCRIPT**
The default style how powers are represented.

justunits.**DEFAULT_DIVIDING_STYLE = UnitDividingStyle.SLASHED**
The default style how unit fractions are represented.

justunits.**DEFAULT_SEPARATION_STYLE = UnitSeparationStyle.DOT**
The default style by which character units are separated.

justunits.**DEFAULT_UNIT_STYLE = UnitPowerStyle.None**
The default unit style composed of power-, dividing- and separation-style.

1.2.2 Seperation of attribute/value and unit

AttributeUnitSeparators.**WHITESPACE_BOXED = 1**
Leads to ‘attribute [unit]’

AttributeUnitSeparators.**UNDERLINE_BOXED = 2**
Leads to ‘attribute_[unit]’

AttributeUnitSeparators.**WHITESPACE_IN = 4**
Leads to ‘attribute in unit’

AttributeUnitSeparators.**UNDERLINED_IN = 8**
Leads to ‘attribute_in_unit’

AttributeUnitSeparators.**HASH_SIGN = 16**
Leads to ‘attribute#unit’

AttributeUnitSeparators.**SINGLE_UNDERLINE = 32**
Leads to ‘attribute_unit’

AttributeUnitSeparators.**SINGLE_WHITESPACE = 64**
Leads to ‘attribute unit’

1.2.3 Styling of just units

Unit power style

`UnitPowerStyle.SUPERSCRIPT = 256`
Leads to superscripting powers like a^1

`UnitPowerStyle.CARET = 512`
Powers are represented in ascii compatible like *a^-1*

Style of divisions

`UnitDividingStyle.SLASHED = 1024`
Units are divided by a forward slash like *a/b*.

`UnitDividingStyle.BY_POWER = 2048`
Divisions of units is represented using powers like *a/b* being ab^1.

Seperation in between units

`UnitSeparationStyle.DOT = 4096`
Units are separated using a dot operator like *ab*

`UnitSeparationStyle.ASTERISK = 8192`
Units are separated by an asterisk like *a*b* being ascii compatible.

Complete unit styles

`UnitStyle.SIMPLE_ASCII = 9728`
Results into 'kg/(s^2*K)*m^2'

`UnitStyle.TOTAL_POWER_ASCII = 10752`
Results into 'kg*s^-2*K^-1*m^2'

`UnitStyle.FINE_SLASHED_SUPERSCRIPT = 5376`
Results into 'kg/(s^2K)m^2'

`UnitStyle.FINE_POWERED_SUPERSCRIPT = 6400`
Results into '$kgs^2K^1m^2$'

1.3 API reference

justunits

justunits.DerivedUnit(*sub_units)
param *sub_units

justunits.AttributeUnitSeparators(value) An enumeration.

justunits.BaseSiUnits()

continues on next page

Table 1 – continued from previous page

justunits.BareUnit(symbol, name, quantity) **param symbol**	
justunits.create_unit(symbol, name, quantity)	Creates a basic unit of your liking.
justunits.from_string(unit_text) **param unit_text**	
justunits.join_unit(attribute, unit[, ...])	Joins attribute and unit using one of six separators.
justunits.Prefix(symbol, name, decimal)	Create new instance of Prefix(symbol, name, decimal)
justunits.prefix_base_unit(base_unit, ...) **param base_unit**	
justunits.reformat_unit(text[, ...])	Reformats a attribute/value-unit pair to a targeted format.
justunits.register_unit(unit)	
justunits.reset_unit_library()	Resets the unit library to the default definition.
justunits.set_default_attribute_unit_separator()	Sets the default separator in between attribute/value and unit.
justunits.set_default_unit_style(unit_style)	Sets the default unit style.
justunits.SiPrefixes()	
justunits.SiUnits()	
justunits.split_unit(text[, ...])	Splits an attribute/value with a trailing unit.
justunits.split_unit_text(text[, ...])	Splits an attribute/value with a trailing unit text.
justunits.to_string(unit[, unit_style])	Creates a string represenation of units.
justunits.UnitDividingStyle(value)	An enumeration.
justunits.UnitPowerStyle(value)	An enumeration.
justunits.UnitSeparationStyle(value)	An enumeration.
justunits.UnitStyle(value)	An enumeration.
justunits.UnknownUnit(unit_text, ...)	The unknown unit appears for cases in which text could not be resolved to units known by the current runtime library.

1.3.1 justunits

Module Attributes

BaseUnit	The **immutable**, minimum unit definition consisting of the unit's symbol, its name and the quantity this units stands for.
PREFIX_FOR_ONE	This is the (non existing prefix) placeholder for 1.
DEFAULT_ATTRIBUTE_UNIT_SEPARATOR	The style separating attributes/values from units.
DEFAULT_ALLOWED_SEPARATORS	By default only 5 of 7 supported attribute unit separator are taken into account as the absolute certain separation of an single underline or whitespace separated unit is ambiguous, without limiting the results to the content of the unit library.

continues on next page

Table 2 – continued from previous page

DEFAULT_POWER_STYLE	The default style how powers are represented.
DEFAULT_DIVIDING_STYLE	The default style how unit fractions are represented.
DEFAULT_SEPARATION_STYLE	The default style by which character units are separated.
DEFAULT_UNIT_STYLE	The default unit style composed of power-, dividing- and separation-style.

Functions

create_unit(symbol, name, quantity[, prefix])	Creates a basic unit of your liking.
from_string(unit_text)	**param unit_text**
join_unit(attribute, unit[, target_format])	Joins attribute and unit using one of six separators.
prefix_base_unit(base_unit, supported_prefixes)	**param base_unit**
reformat_unit(text[, target_format, converter])	Reformats a attribute/value-unit pair to a targeted format.
register_unit(unit)	
reset_unit_library()	Resets the unit library to the default definition.
set_default_attribute_unit_separator(separator)	Sets the default separator in between attribute/value and unit.
set_default_unit_style(unit_style)	Sets the default unit style.
split_attribute_unit(text[, allowed_separators])	**param text**
split_number_and_unit(text)	Explicitly splits a numeric text followed by an unit.
split_unit(text[, allowed_separators, converter])	Splits an attribute/value with a trailing unit.
split_unit_text(text[, allowed_separators])	Splits an attribute/value with a trailing unit text.
to_string(unit[, unit_style])	Creates a string represenation of units.

Classes

AUnit(raw_unit, prefix, power, float] = 1)	A unit represents just a single unit like meter 'm' and second 's'.
AnyUnit(*sub_units)	**param *sub_units**
AttributeUnitSeparators(value)	An enumeration.
BareUnit(symbol, name, quantity)	**param symbol**
BaseSiUnits()	
BaseUnit	The **immutable**, minimum unit definition consisting of the unit's symbol, its name and the quantity this units stands for.

continues on next page

Table 4 – continued from previous page

DerivedUnit(*sub_units)	**param *sub_units**
Prefix(symbol, name, decimal)	Create new instance of Prefix(symbol, name, decimal)
SiPrefixes()	
SiUnits()	
UnitDividingStyle(value)	An enumeration.
UnitPowerStyle(value)	An enumeration.
UnitSeparationStyle(value)	An enumeration.
UnitStyle(value)	An enumeration.
UnknownUnit(unit_text, switch_power, ...)	The unknown unit appears for cases in which text could not be resolved to units known by the current runtime library.

1.3.2 DerivedUnit

class justunits.**DerivedUnit**(**sub_units*)

Parameters

- ***sub_units** –
- **target_style** –

Examples

DerivedUnit are comparable and hashable. The order of units is neglected for the comparison resulting into *ab* == *ba*.

```
>>> import justunits
>>> energy = justunits.from_string("N*m")
>>> ygrene = justunits.from_string("m*N")
>>> energy == ygrene
True
```

DerivedUnit is hashable like *AUnit* to support usage as key within within mappings. In the current implementation the order of internal units is neglected. Meaning *ab* and *ba* are considered being the same.

```
>>> hash(energy) == hash(ygrene)
True
```

```
>>> map = {energy: "energy"}
>>> map[ygrene]
'energy'
```

```
>>> pressure = justunits.from_string("N/m")
>>> pressure != energy
True
```

```
>>> hash(pressure) == hash(energy)
False
```

1.3.3 AttributeUnitSeparators

class justunits.**AttributeUnitSeparators**(*value*)

An enumeration.

1.3.4 BaseSiUnits

class justunits.**BaseSiUnits**

1.3.5 BareUnit

class justunits.**BareUnit**(*symbol: str*, *name: str*, *quantity: str*)

Parameters

- **symbol** –
- **name** –
- **quantity** –

Examples

```
>>> from justunits import BareUnit
>>> pressure = BareUnit("Pa", "pascal", "pressure")
>>> stress = BareUnit("Pa", "pascal", "stress")
>>> pascal = BareUnit("Pa", "pascal", "pressure, stress")
>>> meter = BareUnit("m", "meter", "length")
```

```
>>> pressure == stress == pascal
True
```

```
>>> pascal != meter
True
```

```
>>> hash(pressure) == hash(stress) == hash(pascal)
True
```

```
>>> hash(pascal) != hash(meter)
True
```

```
>>> unit_map = {"m": meter, "Pa": pascal}
>>> unit_map[pascal]
BareUnit('Pa', 'pascal', 'pressure, stress')
```

1.3.6 create_unit

justunits.**create_unit**(*symbol: str*, *name: str*, *quantity: str*, *prefix: Optional[justunits.Prefix] = None*) → *justunits.AUnit*

Creates a basic unit of your liking.

Parameters

- **symbol** – Symbol the carries.
- **name** – The unit's name.
- **quantity** – What the unit stands for.
- **prefix** – An optional, additional prefix.

Returns AUnit

Examples

```
>>> from justunits import create_unit
>>> create_unit("ba", "banana", "fruit", prefix=SiPrefixes.atto)
AUnit(aba 'attobanana' fruit, 1e-18ba)
```

1.3.7 from_string

justunits.**from_string**(*unit_text: str*) → *justunits.DerivedUnit*

Parameters **unit_text** –

Returns:

Examples

```
>>> import justunits
>>> torque = justunits.from_string("Nm")
>>> torque
(AUnit(N 'newton' force, weight), AUnit(m 'meter' length))
```

```
>>> str(torque)
'Nm'
```

```
>>> millisecond = justunits.from_string("ms")
>>> millisecond
(AUnit(ms 'millisecond' time, 0.001s),)
```

```
>>> millisecond.with_prefix(SiPrefixes.nano)
(AUnit(ns 'nanosecond' time, 1e-09s),)
```

```
>>> apple = justunits.from_string("apple")
>>> apple
(UnknownUnit('apple'),)
```

```
>>> justunits.register_unit(create_unit("apple", "apple", "fruit"))
>>> apple = justunits.from_string("apple")
>>> apple
(AUnit(apple 'apple' fruit),)
```

1.3.8 join_unit

justunits.**join_unit**(*attribute: Union[int, float, str]*, *unit: Union[str,* justunits.DerivedUnit, justunits.AUnit, justunits.UnknownUnit*]*, *target_format: Optional[int] = None*) → str

Joins attribute and unit using one of six separators.

Parameters

- **attribute** – The attribute's name which the unit is attached to.
- **unit** – The unit which is being attached.
- **target_format** –

Returns str

Examples

```
>>> import justunits
>>> from justunits import UnitStyle, AttributeUnitSeparators
>>> join_unit(1.23, "apples")
'1.23 [apples]'
```

```
>>> join_unit(1.23, "")
'1.23'
```

```
>>> join_unit(1.23, None)
'1.23'
```

```
>>> join_unit("max. pressure", "MPa", AttributeUnitSeparators.WHITESPACE_IN)
'max. pressure in MPa'
```

```
>>> join_unit("torque", "Nm", AttributeUnitSeparators.UNDERLINED_IN)
'torque_in_Nm'
```

```
>>> join_unit("distance", "mm", AttributeUnitSeparators.WHITESPACE_BOXED)
'distance [mm]'
```

```
>>> join_unit("distance", "mm", AttributeUnitSeparators.HASH_SIGN)
'distance#mm'
```

```
>>> join_unit(1.23, "apples", AttributeUnitSeparators.SINGLE_WHITESPACE)
'1.23 apples'
```

```
>>> join_unit("width", "mm", AttributeUnitSeparators.SINGLE_UNDERLINE)
'width_mm'
```

```
>>> stress = justunits.from_string("N/mm")
>>> stress
(AUnit(N 'newton' force, weight), AUnit(mm 'millimeter' length, 0.001m; power=-1))
>>> join_unit(1.23, stress, AttributeUnitSeparators.WHITESPACE_IN)
'1.23 in N/mm'
```

1.3.9 Prefix

class justunits.**Prefix**(*symbol*, *name*, *decimal*)
Create new instance of Prefix(symbol, name, decimal)

1.3.10 prefix_base_unit

justunits.**prefix_base_unit**(*base_unit: justunits.BareUnit*, *supported_prefixes: Iterable[str]*) → List[*justunits.AUnit*]

Parameters

- **base_unit** –
- **supported_prefixes** –

Returns:

Examples

```
>>> from doctestprinter import doctest_iter_print
>>> base_unit = BareUnit("m", "meter", "length")
>>> units = prefix_base_unit(base_unit, ["m", "k", "µ"])
>>> doctest_iter_print(units, edits_item=lambda x: repr(x))
AUnit(m 'meter' length)
AUnit(mm 'millimeter' length, 0.001m)
AUnit(km 'kilometer' length, 1e+03m)
AUnit(m 'micrometer' length, 1e-06m)
```

```
>>> units = prefix_base_unit(base_unit, "")
>>> doctest_iter_print(units, edits_item=lambda x: repr(x))
AUnit(m 'meter' length)
```

1.3.11 reformat_unit

justunits.**reformat_unit**(*text: str*, *target_format: Optional[enum.IntFlag] = None*, *converter: Optional[Callable[[str], Any]] = None*) → str
Reformats a attribute/value-unit pair to a targeted format. 'None' or missing style definitions are replaced by _defaults.

Parameters

- **text** – Text containing an attribute/value-unit pair.

- **target_format** – The format the attribute/value-unit pair gets. 'None' or not defined styles are set to _defaults.
- **converter** – A callable converting a string into the desired object. A failed conversion must raise a ValueError. Default is str.

Returns str

Examples

```
>>> import justunits
>>> from justunits import reformat_unit, AttributeUnitSeparators, UnitStyle
>>> reformat_unit("1 apple")
'1 [apple]'
>>> hashed = AttributeUnitSeparators.HASH_SIGN
>>> reformat_unit("1 apple", hashed)
'1#apple'
>>> fine_slashed_in_style = (
...     AttributeUnitSeparators.WHITESPACE_IN | UnitStyle.FINE_SLASHED_SUPERSCRIPT
... )
>>> reformat_unit("energy#kg/(m*s^2)")
'energy [kg/(ms²)]'
```

By default single whitespace or underline separated units are not taken into account, allowing the distingtion between any attribute and trailing unit.

```
>>> reformat_unit("without unit")
'without unit'
```

Using a more distinguishable seperator enables the reformatting of *'lazy'* written unit compositions like newton meter being a unit for torque. This is intential as being a method to make clean representations of units.

```
>>> reformat_unit("torque in Nm")
'torque [Nm]'
```

Note: The detection of *'lazy'* written units like 'Nm' is being tested as the default behaviour of just units within the alpha state. You may help by feedbacks onto the github project page https://github.com/david.scheliga/justunits/issues in which cases this leads to problems.

Be aware that this tries for breaking unknown units into known parts. Leading to this such a behavior.

```
>>> reformat_unit("with a unit in unit*pattern")
'with a unit [unitpattern]'
```

```
>>> justunits.from_string("unit")
(AUnit(u 'unified atomic mass unit' ), UnknownUnit('nit'))
```

```
>>> justunits.register_unit(justunits.create_unit("unit", "dummy", "n.a."))
>>> reformat_unit("with a unit in unit*pattern")
'with a unit [unitpattern]'
```

1.3.12 register_unit

justunits.**register_unit**(*unit:* justunits.AUnit)

1.3.13 reset_unit_library

justunits.**reset_unit_library**()
Resets the unit library to the default definition.

1.3.14 set_default_attribute_unit_separator

justunits.**set_default_attribute_unit_separator**(*separator: int*)
Sets the default separator in between attribute/value and unit. The change is limited for the current runtime.

Parameters **separator** – A separator from AttributeUnitSeparators as default.

Raises **ValueError** – If *separator* is not within AttributeUnitSeparators.

Examples

```
>>> import justunits
>>> from justunits import AttributeUnitSeparators
>>> justunits.join_unit("length", "mm")
'length [mm]'
>>> set_default_attribute_unit_separator(AttributeUnitSeparators.WHITESPACE_IN)
>>> justunits.join_unit("length", "mm")
'length in mm'
```

1.3.15 set_default_unit_style

justunits.**set_default_unit_style**(*unit_style: int*)
Sets the default unit style. The change is limited for the current runtime.

Parameters **unit_style** – Combination of UnitPowerStyle, UnitDividingStyle and UnitSeparationStyle.

Raises **ValueError** – If not within *justunits.UnitStyle*.

1.3.16 SiPrefixes

class justunits.**SiPrefixes**

1.3.17 SiUnits

class justunits.**SiUnits**

1.3.18 split_unit

justunits.**split_unit**(*text: str*, *allowed_separators: Optional[*justunits.AttributeUnitSeparators*] = None*, *converter: Optional[Callable[[str], Any]] = None*) → Tuple[str, Union[*justunits.AUnit*, *justunits.UnknownUnit*]]

Splits an attribute/value with a trailing unit.

Parameters

- **text** – Text containing an attribute/value-unit pair.
- **allowed_separators** – The separators which should be taken into account for splitting.
- **converter** – A callable converting a string into the desired object. A failed conversion must raise a ValueError. Default is str.

Returns Tuple[str, Union[AUnit, UnknownUnit]]

Examples

```
>>> import justunits
>>> justunits.split_unit("1 apple", converter=int)
(1, (AUnit(apple 'apple' fruit),))
```

1.3.19 split_unit_text

justunits.**split_unit_text**(*text*, *allowed_separators: Optional[*justunits.AttributeUnitSeparators*] = None*) → Tuple[str, str]

Splits an attribute/value with a trailing unit text.

Parameters

- **text** – Text containing an attribute/value-unit pair.
- **allowed_separators** – The separators which should be taken into account for splitting.

Returns Tuple[Union[str, float], str]

Examples

```
>>> import justunits
>>> justunits.split_unit_text("length in mm")
('length', 'mm')
>>> justunits.split_unit_text("1.23 kN")
('1.23', 'kN')
```

1.3.20 to_string

justunits.**to_string**(*unit: Iterable[*justunits.AUnit*]*, *unit_style: Optional[*justunits.UnitStyle*] = None*) → str
Creates a string represenation of units.

Parameters

- **unit** (*Iterable[*AUnit*]*) – A sequence of units representing a single unit or a composition of units.
- **unit_style** (UnitStyle) – The targeted formatting style to be applied.

Returns str

Examples

```
>>> from justunits import UnitStyle
>>> import justunits
>>> heat_capacity = justunits.from_string("kg/(s^2*K)*m^2")
>>> justunits.to_string(heat_capacity, UnitStyle.FINE_SLASHED_SUPERSCRIPT)
'kg/(s²K)m²'
>>> justunits.to_string(heat_capacity, UnitStyle.FINE_POWERED_SUPERSCRIPT)
'kgs²K¹m²'
>>> justunits.to_string(heat_capacity, UnitStyle.TOTAL_POWER_ASCII)
'kg*s^-2*K^-1*m^2'
>>> justunits.to_string(heat_capacity, UnitStyle.SIMPLE_ASCII)
'kg/(s^2*K)*m^2'
```

1.3.21 UnitDividingStyle

class justunits.**UnitDividingStyle**(*value*)
An enumeration.

1.3.22 UnitPowerStyle

class justunits.**UnitPowerStyle**(*value*)
An enumeration.

1.3.23 UnitSeparationStyle

class justunits.**UnitSeparationStyle**(*value*)
An enumeration.

1.3.24 UnitStyle

class justunits.**UnitStyle**(*value*)
An enumeration.

1.3.25 UnknownUnit

class justunits.**UnknownUnit**(*unit_text: str*, *switch_power: bool = False*, *was_enclosed: bool = False*)
The unknown unit appears for cases in which text could not be resolved to units known by the current runtime library.

Parameters

- **unit_text** – Text for which no 'known' unit could be found.
- **switch_power** – States whether the unit would need to switch power due to division.
- **was_enclosed** – States if the unit text was enclosed by round brackets.

Examples

Fruits are definitivelly unknown to the unit library.

```
>>> import justunits
>>> justunits.from_string("pear/apple")
(UnknownUnit('pear'), UnknownUnit('apple' *^-1))
```

Any unknown occurance can be registered.

```
>>> justunits.register_unit(justunits.create_unit("pear", "pear", "fruit"))
>>> justunits.from_string("pear/apple")
(AUnit(pear 'pear' fruit), UnknownUnit('apple' *^-1))
```

1.3.26 justunits.AUnit

justunits.AUnit(raw_unit, prefix, power, ...)	A unit represents just a single unit like meter 'm' and second 's'.
justunits.AUnit.switch_power()	The unit's power is multiplied by -1.
justunits.AUnit.power	The units power.
justunits.AUnit.width	Width (length) of the unit's symbol (including any prefix).
justunits.AUnit.first_letter	The symbols first letter.
justunits.AUnit.symbol	The unit's symbol including any prefix.
justunits.AUnit.name	The units name.
justunits.AUnit.quantity	The quantity the unit stands for.
justunits.AUnit.base_symbol	The unit's symbol without any prefix.
justunits.AUnit.prefix_symbol	The symbol of the unit's prefix, if carrying one.
justunits.AUnit.with_power(power)	Returns a unit based on this unit's base unit with the requested power.
justunits.AUnit.with_prefix(prefix)	Returns a unit based on this unit's base unit with the requested prefix.

continues on next page

Table 5 – continued from previous page

justunits.AUnit.without_prefix()	The base unit with the current power.
justunits.AUnit.bare()	A unit based on this unit's base unit solely.
justunits.AUnit.to_raw_parts()	Returns:
justunits.AUnit.UNIT_TYPE	

AUnit

class justunits.**AUnit**(*raw_unit: justunits.BareUnit*, *prefix: Optional[*justunits.Prefix*] = None*, *power: Union[int, float] = 1*)

A unit represents just a single unit like meter 'm' and second 's'. In this context a unit is defined a distingtiv symbol (mostly a single letter).

Parameters

- **raw_unit** – The raw unit this unit is based on.
- **prefix** – The SI-Prefix of this unit.
- **power** – Power of this unit. Default is 1.

Examples

A unit is composed of an immutable `RawUnit`, `SiPrefix` and the power the *RawUnit* comes with.

```
>>> from justunits import SiPrefixes
>>> raw_apple = BareUnit("apple", "apple", "fruit")
>>> apple = AUnit(raw_apple)
>>> apple
AUnit(apple 'apple' fruit)
>>> print(apple)
apple
>>> kiloapple = AUnit(raw_apple, prefix=SiPrefixes.kilo)
>>> kiloapple
AUnit(kapple 'kiloapple' fruit, 1e+03apple)
```

AUnit are comparable and hashable to support usage as key within within mappings.

```
>>> another_apple = create_unit("apple", "apple", "fruit")
>>> apple == another_apple
True
```

```
>>> apple != kiloapple
True
```

```
>>> hash(apple) == hash(another_apple)
True
```

```
>>> hash(apple) == hash(kiloapple)
False
```

```
>>> map = {apple: "an apple", kiloapple: "many apples"}
>>> map[kiloapple]
'many apples'
```

switch_power

justunits.AUnit.**switch_power**(*self*)

The unit's power is multiplied by -1.

Examples

```
>>> raw_apple = BareUnit("apple", "apple", "fruit")
>>> sample_unit = AUnit(raw_apple)
>>> str(sample_unit)
'apple'
>>> sample_unit.switch_power()
>>> str(sample_unit)
'apple^-1.0'
```

power

AUnit.**power**

The units power. Default is ^1.

Returns Union[int, float]

width

AUnit.**width**

Width (length) of the unit's symbol (including any prefix).

Returns int

first_letter

AUnit.**first_letter**

The symbols first letter.

Returns str

symbol

AUnit.**symbol**
The unit's symbol including any prefix.

Returns str

name

AUnit.**name**
The units name.

Returns str

quantity

AUnit.**quantity**
The quantity the unit stands for.

Returns str

base_symbol

AUnit.**base_symbol**
The unit's symbol without any prefix.

Returns str

prefix_symbol

AUnit.**prefix_symbol**
The symbol of the unit's prefix, if carrying one.

Notes

The base unit carries a prefix representing 1 with an empty str as prefix.

Returns str

with_power

justunits.AUnit.**with_power**(*self*, *power: Union[int, float]*) → *justunits.AUnit*
Returns a unit based on this unit's base unit with the requested power.

Parameters **power** – Power the new unit should carry.

Returns AUnit

Examples

```
>>> import justunits
>>> from justunits import SiUnits, SiPrefixes
>>> capture_time = SiUnits.second.with_prefix(SiPrefixes.nano)
>>> capture_time
AUnit(ns 'nanosecond' time, 1e-09s)
>>> frames_per_second = capture_time.with_power(-1)
>>> frames_per_second
AUnit(ns 'nanosecond' time, 1e-09s; power=-1)
>>> justunits.to_string(frames_per_second)
'ns'
```

with_prefix

justunits.AUnit.**with_prefix**(*self*, *prefix:* justunits.Prefix) → *justunits.AUnit*

Returns a unit based on this unit's base unit with the requested prefix.

Parameters **prefix** – The prefix of the requested unit.

Returns AUnit

Examples

```
>>> from justunits import SiUnits, SiPrefixes
>>> capture_time = SiUnits.second.with_prefix(SiPrefixes.nano)
>>> capture_time
AUnit(ns 'nanosecond' time, 1e-09s)
```

without_prefix

justunits.AUnit.**without_prefix**(*self*) → *justunits.AUnit*

The base unit with the current power.

Returns AUnit

Examples

```
>>> from justunits import SiUnits, SiPrefixes
>>> capture_time = SiUnits.second.with_prefix(SiPrefixes.nano)
>>> frames_per_second = capture_time.with_power(-1)
>>> frames_per_second
AUnit(ns 'nanosecond' time, 1e-09s; power=-1)
>>> frames_per_second.without_prefix()
AUnit(s 'second' time; power=-1)
```

bare

justunits.AUnit.**bare**(*self*) → *justunits.AUnit*

A unit based on this unit's base unit solely.

Returns AUnit

Examples

```
>>> from justunits import SiUnits, SiPrefixes
>>> capture_time = SiUnits.second.with_prefix(SiPrefixes.nano)
>>> frames_per_second = capture_time.with_power(-1)
>>> frames_per_second
AUnit(ns 'nanosecond' time, 1e-09s; power=-1)
>>> frames_per_second.bare()
AUnit(s 'second' time)
```

to_raw_parts

justunits.AUnit.**to_raw_parts**(*self*) → List

Returns:

Examples

```
>>> from justunits import SiUnits
>>> from doctestprinter import doctest_print
>>> doctest_print(SiUnits.millimeter.to_raw_parts(), max_line_width=50)
[BareUnit('m', 'meter', 'length'), Prefix(symbol='m',
name='milli', decimal=0.001), 1.0]
```

```
>>> doctest_print(SiUnits.meter.to_raw_parts(), max_line_width=50)
[BareUnit('m', 'meter', 'length'), Prefix(symbol='',
name='', decimal=1), 1.0]
```

justunits.AUnit.UNIT_TYPE

AUnit.**UNIT_TYPE = 1**

CHAPTER

TWO

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

INDEX

U

W