bookkeep Documentation

Release 2018

Yoel Cortes-Pena

May 31, 2019

Contents

bookkeep is a python package for keeping track of units of measure and measurement bounds. The package mainly features the SmartBook, a dictionary subclass that incorporates pint Quantity objects for managing units of measure.

CHAPTER 1

SmartBook

class bookkeep.**SmartBook**(*units={}*, *bounds={}*, **args*, *source=None*, ***kwargs*)

Create a dictionary that represents values with units of measure and alerts when setting an item out of bounds. Bounds are always inclusive.

Parameters

units: [UnitManager or dict] Dictionary of units of measure.

bounds: [dict] Dictionary of bounds.

***args:** Key-value pairs to initialize.

source: [str] Should be one of the following [-]:

- Short description of the smartbook.
- Object which the smartbook belongs to.
- None

****kwargs:** Key-value pairs to initialize.

Class Attribute

Quantity: pint Quantity class for compatibility.

Examples

SmartBook objects provide an easy way to keep track of units of measure and enforce bounds.

Create a SmartBook object with *units*, *bounds*, a *source* description, and *arguments* to initialize the dictionary:

```
>>> from bookkeep import SmartBook
>>> sb = SmartBook(units={'T': 'K', 'Duty': 'kJ/hr'},
...                bounds={'T': (0, 1000)},
...                source='Operating conditions',
...                T=350)
```

(continues on next page)

(continued from previous page)

```
>>> sb
{'T': 350 (K)}
```

The *units* attribute becomes a *UnitManager* object with a reference to all dictionaries (*clients*) it controls. These include the SmartBook and its bounds.

```
>>> sb.units
UnitManager:
{'T': 'K',
 'Duty': 'kJ/hr'}
>>> sb.units.clients
[{'T': 350 (K)}, {'T': (0, 1000)}]
```

Change units:

```
>>> sb.units['T'] = 'degC'
>>> sb
{'T': 76.85 (degC)}
>>> sb.bounds
{'T': (-273.15, 726.85)}
```

Add items:

```
>>> sb['P'] = 101325
>>> sb
{'T': 76.85 (degC),
 'P': 101325}
```

Add units:

```
>>> sb.units['P'] = 'Pa'
>>> sb
{'T': 76.85 (degC),
 'P': 101325 (Pa)}
```

A BookkeepWarning is issued when a value is set out of bounds:

```
>>> sb['T'] = -300
__main__:1: BookkeepWarning: @Operating conditions: T (-300 degC) is out␣
↪of bounds (-273.15 to 726.85 degC).
```

Nested SmartBook objects are easy to read, and can be made using the same units and bounds.

Create new SmartBook objects:

```
>>> sb1 = SmartBook(sb.units, sb.bounds,
...                 T=25, P=500)
>>> sb2 = SmartBook(sb.units, sb.bounds,
...                 T=50, Duty=50)
>>> sb1
{'T': 25 (degC),
 'P': 500 (Pa)}
>>> sb2
{'T': 50 (degC),
 'Duty': 50 (kJ/hr)})
```

Create a nested SmartBook object:

```
>>> nsb = SmartBook(units=sb.units, sb1=sb1, sb2=sb2)
>>> nsb
{'sb1':
    {'T': 25 (degC),
     'P': 500 (Pa)},
 'sb2':
    {'T': 50 (degC),
     'Duty': 50 (kg/hr)}}
```

pint Quantity objects are also compatible, so long as the corresponding Quantity class is set as the Quantity attribute.

Set a Quantity object:

```
>>> Q_ = SmartBook.Quantity
>>> sb1.bounds['T'] = Q_((0, 1000), 'K')
>>> sb1['T'] = Q_(100, 'K')
>>> sb1
{'T': -173.15 degC,
 'P': 500 (Pa)}
```

Setting a Quantity object out of bounds will issue a warning:

```
>>> sb1['T'] = Q_(-1, 'K')
 __main__:1: BookkeepWarning: T (-274.15 degC) is out of bounds (-273.15␣
↪to 726.85 degC).
```

Trying to set a Quantity object with wrong dimensions will raise an error:

```
>>> Q_ = SmartBook.Quantity
>>> sb1['T'] = Q_(100, 'meter')
DimensionalityError: Cannot convert from 'meter' ([length]) to 'degC'␣
↪([temperature])
```

`class Quantity`

`bounds`
Dictionary of bounds.

`boundscheck`(*key*, *value*)
Return True if value is within bounds. Return False if value is out of bounds and issue a warning.

Parameters

key: [str] Name of value

value: [number, Quantity, or array]

`classmethod enforce_boundscheck`(*val*)
If *val* is True, issue BookkeepWarning whenever an item is set out of bounds. If *val* is False, ignore bounds.

`classmethod enforce_unitscheck`(*val*)
If *val* is True, adjust Quantity objects to correct units. If *val* is False, ignore units.

`nested_items`()
Return all key-value pairs of self and nested SmartBook objects.

`nested_keys`()
Return all keys of self and nested SmartBook objects.

`nested_values``()`
Return all values of self and nested SmartBook objects.

`source`
Short description or object it describes

`units`
Dictionary of units of measure.

`unitscheck``(`*key*, *value*`)`
Adjust Quantity objects to correct units and return True.

CHAPTER 2

UnitManager

class bookkeep.**UnitManager**(*clients*, **args*, ***kwargs*)

Create a UnitManager object for handling units of measure of a list of dictionaries (*clients*). When an item in UnitManger changes, all dictionaries in *clients* with the same key change values accordingly.

Parameters

clients: [list] All dictionaries managed by UnitManager object.

***args:** Key/units pairs.

****kwargs:** Key/units pairs.

Class Attribute

Quantity: pint Quantity class for compatibility.

Examples

Convert units of all clients using a UnitManager.

Create client dictionaries:

```
>>> car = {'weight': 4000, 'velocity': 50}
>>> plane = {'weight': 175000, 'velocity': 600}
```

Create a UnitManager object:

```
>>> um = UnitManager([car, plane], weight='lbs', velocity='mph')
>>> um
UnitManager:
{'weight': 'lbs',
 'velocity': 'mph'}
```

Change units of clients:

```
>>> um['weight'] = 'kg'
>>> um['velocity'] = 'km/hr'
```

(continues on next page)

(continued from previous page)

```
>>> car
{'weight': 1814.36948, 'velocity': 80.46719999999999}
>>> plane
{'weight': 79378.66475000001, 'velocity': 965.6063999999999}
```

Quantity objects are also compatible with UnitManager objects, so long as they are set as the "Quantity" class attribute.

Set "Quantity" attribute:

```
>>> from pint import UnitRegistry
>>> ureg = UnitRegistry()
>>> UnitManager.Quantity = Q_ = ureg.Quantity
```

Set a Quantity object and change units:

```
>>> car['weight'] = Q_(4000, 'lb')
>>> um['weight'] = 'kg'
>>> car
{'weight': <Quantity(1814.36948, 'kilogram')>,
 'velocity': 80.46719999999999>}
```

`class Quantity`

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index