Python

Contents

Easily use, manipulate, and store unit-aware measurement objects using Python and Django.

Contents

Contents:

1.1 Using Measurement Objects in Forms

This is an example for a simple form field usage:

```python
from django import forms
from django_measurement.forms import MeasurementField

class BeerForm(forms.Form):

    volume = MeasurementField(Volume)
```

You can limit the units in the select field by using the 'unit_choices' keyword argument. To limit the value choices of the MeasurementField uses the regular 'choices' keyword argument:

```python
class BeerForm(forms.Form):

    volume = MeasurementField(
        measurement=Volume,
        unit_choices=(("l","l"), ("oz","oz")),
        choices=((1.0, 'one'), (2.0, 'two'))
    )
```

If unicode symbols are needed in the labels for a MeasurementField, define a LABELS dictionary for your subclassed MeasureBase object:

```python
# -*- coding: utf-8 -*-
from sympy import S, Symbol

class Temperature(MeasureBase):
    SU = Symbol('kelvin')
    STANDARD_UNIT = 'k'
    UNITS = {
        'c': SU - S(273.15),
```

```python
        'f': (SU - S(273.15)) * S('9/5') + 32,
        'k': 1.0
    }
    LABELS = {
        'c':u'°C',
        'f':u'°F',
        'k':u'°K',
    }
```

For a *MeasurementField* that represents a *BidimensionalMeasure*, you can set the separator either in settings.py (*MEA-SUREMENT_BIDIMENSIONAL_SEPARATOR is '/'* by default, add setting to override for all BiDimensionalMeasure subclasses) or override for an individual field with the kwarg bidimensional_separator:

```python
speed = MeasurementField(
    measurement=Speed,
    bidimensional_separator=' per '
)

# Rendered option labels will now be in the format "ft per s", "m per hr", etc
```

1.2 Installation

You can either install from pip:

```
pip install django-measurement
```

or checkout and install the source from the [github repository](#):

```
git clone https://github.com/coddingtonbear/django-measurement.git
cd django-measurement
python setup.py install
```

1.3 Measures

See [python-measurement's documentation](#) for information about what measures are available.

1.4 Settings

1.4.1 `MEASUREMENT_BIDIMENSIONAL_SEPARATOR`

For any BidimensionalMeasure, what is placed between the primary and reference dimensions on rendered label

    MEASUREMENT_BIDIMENSIONAL_SEPARATOR = " per "

Defaults to "/". Can be overriden as kwarg *bidimensional_separator* for a given MeasurementField.

1.5 Storing Measurement Objects

Suppose you were trying to cut back on drinking, and needed to store a log of how much beer you drink day-to-day; you might (naively) create a model like such:

```python
from django_measurement.models import MeasurementField
from measurement.measures import Volume
from django.db import models


class BeerConsumptionLogEntry(models.Model):
    name = models.CharField(max_length=255)
    volume = MeasurementField(measurement=Volume)

    def __str__(self):
        return '%s of %s' % (self.name, self.volume)
```

and assume you had a pint of Ninkasi's Total Domination; you'd add it to your log like so:

```python
from measurement.measures import Volume

beer = BeerConsumptionLogEntry()
beer.name = 'Total Domination'
beer.volume = Volume(us_pint=1)
beer.save()

print beer # '1 us_pint of Total Domination'
```

Perhaps you next recklessly dove into your stash of terrible, but nostalgia-inducing Russian beer and had a half-liter of Baltika's #9; you'd add it to your log like so:

```python
another_beer = BeerConsumptionLogEntry()
another_beer.name = '#9'
another_beer.volume = Volume(l=0.5)
another_beer.save()

print beer # '0.5 l of #9'
```

Note that although the original unit specified is stored for display, that the unit is abstracted to the measure's standard unit for storage and comparison:

```python
print beer.volume                     # '1 us_pint'
print another_beer.volume             # '0.5 l'
print beer.volume > another_beer.volume # False
```

1.5.1 How is this data stored?

Since django-measurement v2.0 there value will be stored in a single float field.

1.6 Using Measurement Objects

You can import any of the above measures from *measurement.measures* and use it for easily handling measurements like so:

```python
from measurement.measures import Weight

w = Weight(lb=135)  # Represents 135lbs
print w             # '135.0 lb'
print w.kg          # '61.234919999999995'
```

See Python-measurement's documentation for more information about interacting with measurements.

CHAPTER 2

Summary:

[django.contrib.gis.measure](#) has these wonderful 'Distance' objects that can be used not only for storing a unit-aware distance measurement, but also for converting between different units and adding/subtracting these objects from one another.

This module provides for a django model field and admin interface for storing any measurements provided by [python-measurement](#).

Example use with a model:

```python
from django_measurement.models import MeasurementField
from measurement.measures import Volume
from django.db import models

class BeerConsumptionLogEntry(models.Model):
    name = models.CharField(max_length=255)
    volume = MeasurementField(measurement=Volume)

    def __unicode__(self):
        return u"%s of %s" % (self.name, self.volume)

entry = BeerConsumptionLogEntry()
entry.name = 'Bear Republic Racer 5'
entry.volume = Volume(us_pint=1)
entry.save()
```

These stored measurement objects can be used in all of the usual ways supported by [python-measurement](#) too:

```python
>>> from measurement.measures import Weight
>>> weight_1 = Weight(lb=125)
>>> weight_2 = Weight(kg=40)
>>> added_together = weight_1 + weight_2
>>> added_together
Weight(lb=213.184976807)
>>> added_together.kg  # Maybe I actually need this value in kg?
96.699
```

- Documentation for django-measurement is available via Read the Docs.

- Please post issues on GitHub.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search