Django Basemix Documentation

Release 0.1

Steve Kossouho

Jun 13, 2017

Contents:

Django Basemix is a library of useful utilities and mixins you do not find in Django by default, and might otherwise be scattered in other libraries.

CHAPTER 1

Model mixins

Geometry

class basemix.mixins.geometry.rectangle.**DimensionBase**(**args*, ***kwargs*)

Base class for models with a representation of a rectangle

Classes inheriting from this base model must add two fields named `width` and `height` that must resolve to a type than can be parts of arithmetic operations.

Two model mixins *DimensionIntBase* and *DimensionFloatBase* already exist for our basic needs.

get_area()

Gets the area of the rectangle

Returns Product of the width and height of the rectangle

Return type int or float

get_perimeter()

Gets the perimeter of the rectangle

Returns Twice the sum of height and width

Return type int or float

is_point()

Returns if the rectangle is actually a point

Returns True if width and height are zero

Return type bool

is_segment()

Returns if the rectangle is actually a segment, but not a point

Returns True if only one of width and height is zero

Return type bool

set_dimension(*width*, *height*)
Sets the dimensions of the rectangle

Parameters

- **width** (*int or float*) – width of the rectangle
- **height** (*int or float*) – height of the rectangle

Returns True if changed, False otherwise

class basemix.mixins.geometry.rectangle.**DimensionFloatBase**(**args*, ***kwargs*)
Base class for models with a representation of rectangle dimensions

The model allows for representation of floating point dimensions and contains two `FloatField` named `width` and `height`.

Attributes:

width float width

height float height

class basemix.mixins.geometry.rectangle.**DimensionIntBase**(**args*, ***kwargs*)
Base class for models with a representation of rectangle dimensions

The model allows for representation of integer dimensions and contains two `PositiveIntegerField` named `width` and `height`.

Attributes:

width integer width

height integer height

Date and time

class basemix.mixins.datetime.creation.**CreationBase**(**args*, ***kwargs*)
Base class for models with a creation date

The mixin provides a set of handy methods to manage the status of the creation date. By default, the field is not setup to have an index defined on it. However, in Django 1.11, you can define an index on the field `creation_date`

Attributes:

creation_date date of creation of the object

is_new(*hours=168*)
Returns whether this instance was created recently

Parameters **hours** – Minimum hours to consider the instance not new anymore. Default value is one week (*168*).

Returns `True` if the instance is recent enough

Return type bool

class basemix.mixins.datetime.update.**UpdateBase**(**args*, ***kwargs*)
Base class for models with an update date

The mixin provides a set of handy methods to manage the status of the update time. By default, the field is not setup to have an index defined on it. However, in Django 1.11, you can define an index on the field `update_date`

is_updated_recently(*hours=168*)
Returns whether this instance was updated recently

Parameters **hours** – Minimum hours to consider the instance not fresh anymore. Default value is one week (*168*).

Returns `True` if the instance update time is recent enough

Return type bool

Priority and weight

class basemix.mixins.priority.priority.**PriorityBase**(**args*, ***kwargs*)
Base class for models that can be ordered by priority

The mixin adds a new field named `priority`, which is an integer between *0* and *100* (enforced by field type and a validator)

Contrary to the UNIX nonsense where a high nice priority is -20 and a low nice priority is 19, we use the intuitive notion of priority here: a high number is akin to a high priority.

Attributes:

priority priority of the object as an integer, `0..100`

save(**args*, ***kwargs*)
Save the object to the database

Content

class basemix.mixins.content.content.**ContentBase**(**args*, ***kwargs*)
Base class for models that share content attributes

The attributes added by this mixin are `title`, `description`, `content` and `is_visible`.

Attributes:

is_visible whether the content should be displayed by normal users

title title of the content, at most 192 characters

description most content objects have a description, with an unlimited size

content actual content of the object, with an unlimited size

save(**args*, ***kwargs*)
Save the object to the database

Switches

class basemix.mixins.content.switches.**Active**(**args*, ***kwargs*)
Base class for models that can be enabled/disabled

This mixin adds one field named `is_active`

Attributes:

is_active defines whether the item is active or not. Non-nullable

class basemix.mixins.content.switches.**Visible**(**args*, ***kwargs*)

Base class for models with a visibility field

This mixin adds one field named `is_visible`

Attributes:

is_visible visibility of the item. Non-nullable

CHAPTER 2

Utilities

Decorators

basemix.utility.decorators.**addattr**(***kwargs*)

Decorator to add attributes to a function

The shortcut is most useful for admin representations of methods or attributes.

Example:

Instead of writing

```
>>> def is_valid(self):
>>>     return self.name != "foo"
>>> is_valid.short_description = "The name for the function"
>>> is_valid.boolean = True
```

You write

```
>>> @addattr(short_description="The name for the function", boolean=True)
>>> def is_valid(self):
>>>     return self.name != "foo"
```

Parameters **kwargs** – the properties to add to a function

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index