ObjectRelationMixin, a generator for specific abstract base classes

# History

|  |  |
| --- | --- |
| Created by | Rupert |
| Created | 28-07-2008 |
| Last modified by | - |
| Last modified | - |

# Introduction

There is a new abstract class ObjectRelationMixin (generator) available in the jetson project, which can be used as a mixin for any model. It describes a fragment for dealing with Django’s content types – object ids (see [1]). It implements generic foreign keys to any object with some useful methods and validators for the admin. It also uses dynamic fields, so it can be used to attach more than one generic foreign key to a model using the same mixin. It is implemented as a class generator function. For details about model inheritance and using mixin classes, see [2], [3]

# Example Usage

## Simple Example

Suppose, you want to create a simple model Blog with a generic foreign key and a title. The generic foreign key is used to relate an instance of the model to any object. With the new ObjectRelationMixin generator, the model would look like this:

class **Blog**(ObjectRelationMixin()):

title = models.CharField(\_(*'title'*), max\_length=255)

class **Admin**:

list\_display = (*'title'*,) + ObjectRelationMixin().get\_fields()

## Advanced Example

Our Blog model should be extended by a second generic foreign key to relate to another object. To avoid clashes in the model fields, we can define our blog model like this:

class **Blog**(ObjectRelationMixin(*"obj1"*, *"Object 1"*, True),

ObjectRelationMixin(*"obj2"*, *"Object 2"*, True)):

title = models.CharField(\_(*'title'*), max\_length=255)

class **Admin**:

list\_display = (*'title'*,) +\

ObjectRelationMixin(*"obj1"*, *"Object 1"*, True).get\_fields() + \

ObjectRelationMixin(*"obj2"*, *"Object 2"*, True).get\_fields()

With some additional parameters in the function call, the content type – object id fields get unique names.

# Usage

The ObjectRelationMixin function is called with two optional parameters:

def **ObjectRelationMixin**(prefix=None, prefix\_verbose=None, add\_related\_name=False)

The function generates a base class with some dynamic fields and methods. The generated class is designed for usage in mixin classes. The generated class is abstract, so only subclasses can be instantiated. (For details about abstract base classes and model inheritance in Django, see [2]). The created fields are

|  |  |
| --- | --- |
| <<prefix>>\_content\_type | Field name for the "content type" |
| <<prefix>>\_object\_id | Field name for the "object Id" |
| <<prefix>>\_content\_object | Field name for the "content object" |

with a prefix, which is given as function parameter. There is also a method attached, which returns the content object defined by content type and object id:

get\_<<<<prefix>>\_content\_object>>()

The function parameters are as follows:

## prefix

A prefix, which is added in front of the fields to make them unique. If you use, for instance, “foo” as prefix, the generated fields will be

foo\_content\_type, foo\_object\_id, foo\_content\_object

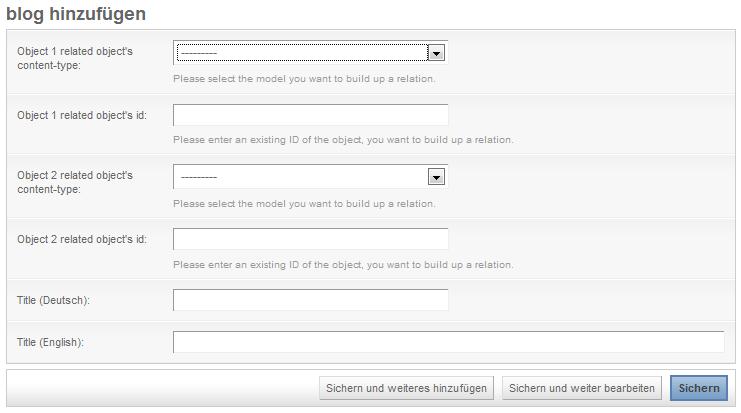
## prefix\_verbose

In the admin, the prefix\_verbose parameter is used for displaying the model fields like this:

|  |  |
| --- | --- |
| <<prefix>>\_content\_type | is displayed as “<< prefix\_verbose>> related object’s content type” |
| <<prefix>>\_object\_id | is displayed as “<< prefix\_verbose>> related object’s id” |
| <<prefix>>\_content\_object | is displayed as “<< prefix\_verbose>> Content Object” |

Using “Object 1” resp. “Object 2” as prefix\_verbose, the Admin interface would look like this:





## add\_related\_name

a Boolean value indicating, that a related name for the generated content type foreign key should be added. This value should be true, if you use more than one ObjectRelationMixin in your model.

# References

|  |  |
| --- | --- |
|  | „Django Documentation”, Chapter 34, “Generic Relations”, Django Software Foundation, 2005-2008, available via http://www.djangoproject.com/documentation/models/generic\_relations/ |
|  | „Django Documentation”, “Model Inheritcance”, Django Software Foundation, 2005-2008, available via http://code.djangoproject.com/wiki/ModelInheritance |
|  | “Eric Florenzano’s Blog”, “[Exploring Mixins with Django Model Inheritance](http://www.eflorenzano.com/blog/post/exploring-mixins-django-model-inheritance/)”, Eric Florenzano, 2007, available via http://www.eflorenzano.com/blog/post/exploring-mixins-django-model-inheritance/ |