A generic URL-Object-Mapping mechanism (for Studio 38, Project Jetson)

# History

|  |  |
| --- | --- |
| Created by | Rupert |
| Created | 12-01-2009 |
| Last modified by | Rupert |
| Last modified | 13-01-2009 |

# Introduction

As the Jetson project becomes a CMS more and more, a generic mechanism for mapping URLs to specific Django objects is needed. Suppose you have a blog application, which can be “hooked” to any object. In your web application, any registered user should get his own blog, where he can post his ideas. For example, the blog for the user “Aidas” should be available via the URL

[www.example.com/person/aidas/blog](http://www.example.com/person/aidas/blog).

In this case, the URL part “person/aidas” would identify the “object” ( the user Aidas), the last part “blog” would determine the application (the blog application for the user Aidas is requested).

Currently, the Django urls.py must be modified for each model (that is the user in our example case, but it also can be any other model), a blog application can be attached to. For instance, if the blog app should be attached to institutions, groups and persons, three entries in the urls.py would be needed. You would write something like

…  
url(r*'institution/(?P<slug>[^/]+)/blog/)'*, include(*'.apps.blog.urls'*),  
url(r*'person/(?P<slug>[^/]+)/blog/)'*, include(*'.apps.blog.urls'*),  
url(r*'group/(?P<slug>[^/]+)/blog/)'*, include(*'.apps.blog.urls'*),

…

This task can be simplified and generalized. If you just need to write

url(r*'(?P<object\_url\_part>.\*)blog/'* include(*'general.apps.blog.urls'*), {

*'included'* : [*'person', 'institution', 'group'*],

}.

Or you want to allow blogs for every object except Institutions. In this case, you would write

url(r*'(?P<object\_url\_part>.\*)blog/'* include(*'general.apps.blog.urls'*), {

*'excluded* : [*'institution'*],

}.

Everything would go simpler, wouldn’t it? So, applying this generic mechanism would allow to hook a Django application to any object. Adding specific URLs is no longer needed.

# A Generic Object Mapper for URLs

The mechanism described above needs some “mapping” of URLs to objects. Every object must be uniquely identified by a URL. In the example above, the object “Person Aidas” would be uniquely identified by the URL part “person/aidas”. So we need some “mapping” of URLs to Django models. This is done by a special dictionary defined in the urls.py:

OBJECT\_URL\_MAPPER = {

URL\_ID\_PERSON: (Person, *'user\_\_username'*, *"people/details\_base.html"*),

URL\_ID\_INSTITUTION:(Institution, *'slug'*, *"institutions/people/details\_base.html"*),

URL\_ID\_DOCUMENT: (Document, *'slug'*, *"resources/documents/details\_base.html"*),

URL\_ID\_EVENT: (Event, *'slug'*, *"events/details\_base.html"*),

URL\_ID\_PERSONGROUP:(PersonGroup, *'slug'*, *"groups\_networks/groups/details\_base.html"*),

}

The key of the dictionary is some „URL identifier” of the model (URL\_ID\_PERSON = „person“ in the first case), the values are a tuple consisting of the mapped model (in the first case the Person model) and some Django styled field name for looking up the appropriate object in the model. Additionally, a default “base template” is specified. This is used for writing generalized templates. In the second case above, an Institution object identified by URL\_ID\_INSTITUTION (that is “institution”, for example), which his mapped to the Institutions model looks in the “slug” field to get the requested object. The first entry is a little bit more complicated: To get the requested person from the Person model, You will look for a Foreign key to the User model, where the desired Name can be found in the “username” field.

The tuple can be expanded to any other information, maybe the name of default template directories or anything else.

# Implementation

To get an object by passing some URL part, we just need a generic function get\_object\_from\_url. This function can be implemented as follows:

def **get\_object\_from\_url**(object\_url\_part, \*\*kwargs):

*"""*

*tries to identify a related object from an "objects url part".*

*An url part is formed like*

*"<<model\_identifier>>/<<object\_identifier>>",*

*for example "person/aidas". In this case, "person" would*

*be an identifier for the "Person" model and "Aidas" would be*

*the "object identifier". Here, it denotes*

*the username name of Aidas.*

*kwargs can optionally have two additional key, value pairs:*

*('exclude', << a list of model\_identifiers to exclude >>)*

*('include', << a list of model\_identifiers to include >>)*

*exclusion and inclusion may not be used together at the moment.*

*Return value is the object or None (if it cannot be identified*

*or the URL part is missing)*

*"""*

obj = None

if object\_url\_part is None or object\_url\_part == *""*:

(model\_identifier, object\_identifier) = (None, None)

else:

(model\_identifier, object\_identifier) = object\_url\_part.strip(*'/'*).split(*'/'*)

try:

object\_url\_mapper = getattr(\_\_import\_\_(settings.ROOT\_URLCONF, {}, {}, [*''*]),

*'OBJECT\_URL\_MAPPER'*)

except (ImportError, AttributeError):

raise Http404, *"Please specify an OBJECT\_URL\_MAPPER dict to use this function"*

if object\_url\_mapper.has\_key(model\_identifier):

object\_props = object\_url\_mapper[model\_identifier]

try:

obj = object\_props[0].objects.get(\*\*{object\_props[1]:object\_identifier})

except:

raise Http404, *"Sorry, requested object '%s' does not exist in the %s model"* %

(object\_identifier, str(object\_props[0]))

# now test, if model is supported and allowed!

if kwargs.has\_key(*'include'*):

if not model\_identifier in kwargs[*'include'*]:

raise Http404, *"Sorry, you are not allowed to access object '%s' in the requested*

*application"* % (object\_identifier)

if kwargs.has\_key(*'exclude'*):

if model\_identifier in kwargs[*'exclude'*]:

raise Http404, *"Sorry, you are not allowed to access object '%s' in the requested*

*application"* % (object\_identifier)

return obj