django-machina Documentation

Release 0.1.0

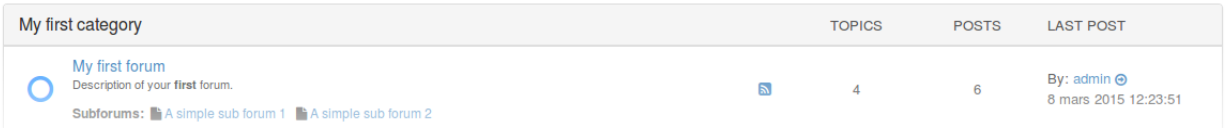
Morgan Aubert

February 28, 2016

Contents

Django-machina is a forum engine for Django providing a way to build community-driven websites. It offers a full-featured yet very extensible forum solution that is designed to be used inside existing Django applications.

Django-machina is customizable and extensible: each single functionality of the application can be customized or overriden to accommodate with your needs and your own business logic. The central aim of *django-machina* is to provide a solid core of a forum project - without much of extra functionality included - that can be extended or customized to suit your project needs.



CHAPTER 1

Features

- Forums tree management
- Per-forum permissions
- Topic and post editing
- Anonymous posting
- Pre-moderation and moderation
- Polls and attachments

CHAPTER 2

Using django-machina

2.1 Getting started

2.1.1 Requirements

- Python 2.7, 3.3, 3.4 or 3.5
- Django 1.8.x or 1.9.x
- Pillow 2.2. or higher
- Django-model-utils 2.0. or higher
- Django-mptt 0.8. or higher
- Django-haystack 2.1. or higher
- Django-markdown 0.7. or higher
- Django-widget-tweaks 1.4. or higher

Note: *Django-machina* uses Markdown (*django-markdown*) by default as a syntax for forum messages, but you can change this in your settings.

2.1.2 Installation

Install *django-machina* using:

```
pip install django-machina
```

Note: Please remember that *django-machina* is currently in alpha. It is not yet suitable for production environments.

2.1.3 Project configuration

Django settings

First update your `INSTALLED_APPS` in your project's settings module. Modify it to be a list and append the *django-machina*'s apps to this list:

```
from machina import get_apps as get_machina_apps

INSTALLED_APS = [
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.sites',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'django.contrib.admin',

  # Machina related apps:
  'mptt',
  'haystack',
  'widget_tweaks',
  'django_markdown',
] + get_machina_apps()
```

Note: As previously stated, Markdown is the default syntax used for forum messages.

Django-machina uses *django-mptt* to handle the tree of forum instances. Search capabilities are provided by *django-haystack*.

Then update your `TEMPLATE_CONTEXT_PROCESSORS` setting as follows:

```
TEMPLATE_CONTEXT_PROCESSORS = (
  # ...
  # Machina
  'machina.core.context_processors.metadata',
)
```

Next add the `machina.apps.forum_permission.middleware.ForumPermissionMiddleware` to your `MIDDLEWARE_CLASSES` setting:

```
MIDDLEWARE_CLASSES = (
    # ...
    # Machina
    'machina.apps.forum_permission.middleware.ForumPermissionMiddleware',
)
```

Then edit your `TEMPLATE_DIRS` setting so that it includes the *django-machina*'s template directory:

```
from machina import MACHINA_MAIN_TEMPLATE_DIR

TEMPLATE_DIRS = (
  # ...
  MACHINA_MAIN_TEMPLATE_DIR,
)
```

Then edit your `STATICFILES_DIRS` setting so that it includes the *django-machina*'s static directory:

```
from machina import MACHINA_MAIN_STATIC_DIR

STATICFILES_DIRS = (
  # ...
  MACHINA_MAIN_STATIC_DIR,
)
```

Finally you have to add a new cache to your settings. This cache will be used to store temporary post attachments. Note that this `machina_attachments` cache must use the `django.core.cache.backends.filebased.FileBasedCache` backend, as follows:

```
CACHES = {
  'default': {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
  },
  'machina_attachments': {
    'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
    'LOCATION': '/tmp',
  }
}
```

Django-haystack settings

Django-machina uses *django-haystack* to provide search for forum conversations. *Django-haystack* allows you to plug in many search backends so you may want to choose the one that best suits your need.

You can start using the basic search provided by the *django-haystack*'s simple backend:

```
HAYSTACK_CONNECTIONS = {
  'default': {
    'ENGINE': 'haystack.backends.simple_backend.SimpleEngine',
  },
}
```

You can also decide to use a more powerfull backend such as *Solr* or *Whoosh*:

```
HAYSTACK_CONNECTIONS = {
  'default': {
    'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
    'PATH': os.path.join(PROJECT_PATH, 'whoosh_index'),
  },
}
```

2.1.4 Database and migrations

Django-machina only provides new-style migrations. So if you are using Django 1.8 or higher, just use the `syncdb` or `migrate` commands:

```
python manage.py migrate
```

2.1.5 URLs configuration

Finally you have to update your main `urls.py` module in order to include the forum' URLs and the *django-markdown*' URLs:

```
from machina.app import board

urlpatterns = patterns(
  # [...]

  # Apps
  url(r'^markdown/', include( 'django_markdown.urls')),
```

```
  url(r'^forum/', include(board.urls)),
)
```

2.1.6 Creating your first forums

You can now navigate to http://127.0.0.1:8000/forum/ in order to visualize the index of your forum board. As you should see no forum have been created yet. *Django-machina* does not ship with pre-created forums, so you should navigate to your administration panel and create some forum instances.

Note: A common practice when creating forums is to embed them in categories in order to better organize the tree of forum instances. Please refer to Glossary if you do not know what a category is in a forum tree.

Congrats! You're in.

2.2 Example projects

Django-machina provides two example projects:

- a "vanilla" project which contains a standard installation of *django-machina* without customization
- a "demo" project which showcases the customization possibilities of *django-machina* (templates, logic, etc)

2.2.1 The vanilla project

The "vanilla" project contains a minimum installation of *django-machina* where no customizations have been made. The project uses the default forum settings and can be usefull for discovering *django-machina*'s functionnalities.

To run this project locally, you can follow these instructions:

```
$ git clone https://github.com/ellmetha/django-machina
$ cd django-machina
$ mkvirtualenv machina_vanilla_project
(machina_vanilla_project) $ make install && pip install -r example_projects/vanilla/requirements.txt
(machina_vanilla_project) $ cd example_projects/vanilla/src/
(machina_vanilla_project) $ python manage.py migrate
(machina_vanilla_project) $ python manage.py createsuperuser
(machina_vanilla_project) $ python manage.py loaddata fixtures/*
(machina_vanilla_project) $ python manage.py runserver
```

Note: The previous steps assume you have Virtualenvwrapper installed on your system.

2.2.2 The demo project

The "demo" project aims to show the possibilities of *django-machina* in terms of personalization and customization. It showcases how *django-machina* can be used to integrate a forum into a Django project. Some of the customisations that are included in this "demo" project are listed bellow:

- a new theme
- the use of django-ckeditor instead of django-markdown

To run this project locally, you can follow these instructions:

```
$ git clone https://github.com/ellmetha/django-machina
$ cd django-machina
$ mkvirtualenv machina_demo_project
(machina_demo_project) $ make install && pip install -r example_projects/demo/requirements.txt
(machina_demo_project) $ cd example_projects/demo/src/
(machina_demo_project) $ python manage.py migrate
(machina_demo_project) $ python manage.py createsuperuser
(machina_demo_project) $ python manage.py loaddata fixtures/*
(machina_demo_project) $ python manage.py runserver
```

2.3 Settings

This is a comprehensive list of all the settings *django-machina* provides. All settings are optional.

2.3.1 General

MACHINA_FORUM_NAME

Default: `'Machina'`

The forum name.

MACHINA_MARKUP_LANGUAGE

Default: `('django_markdown.utils.markdown', {})`

This setting defines how posts content is translated into HTML on the forum. It should be a two-tuple. The first element should be a string corresponding to the Python dotted path to a function returning HTML from a content expressed in a markup language. The second element of the tuple is a dictionary of keyword arguments to pass to the latest function (the dictionary should be empty if the function does not require any argument). Note that if you do not want to use a markup language such as Markdown or BBCode (eg. if you are using a Wysiwyg editor), you can set this setting to `None`.

Django-machina uses Markdown as the default syntax for forum messages.

MACHINA_MARKUP_WIDGET

Default: `'django_markdown.widgets.MarkdownWidget'`

This setting defines the widget used inside topic and post forms. It should be a Python dotted path to a Django form widget.

2.3.2 Forum

MACHINA_FORUM_IMAGE_UPLOAD_TO

Default: `'machina/forum_images'`

The media subdirectory where forum images should be uploaded.

MACHINA_FORUM_IMAGE_WIDTH

Default: `100`

The width used to create the thumbnail that is displayed for each forum that has an image in the list of forums. The image is not resized if this setting is set to `None`.

MACHINA_FORUM_IMAGE_HEIGHT

Default: `70`

The height used to create the thumbnail that is displayed for each forum that has an image in the list of forums. The image is not resized if this setting is set to `None`.

MACHINA_FORUM_TOPICS_NUMBER_PER_PAGE

Default: `20`

The number of topics displayed inside one page of a forum.

2.3.3 Conversation

MACHINA_TOPIC_ANSWER_SUBJECT_PREFIX

Default: `'Re:'`

This is the prefix used to pre-populate the subject of a topic reply. For example: if a reply is being posted for the *Lorem Ipsum* topic, the prefilled subject will be *Re: Lorem Ipsum* in the reply form.

MACHINA_TOPIC_POSTS_NUMBER_PER_PAGE

Default: `15`

The number of posts displayed inside one page of a forum topic.

MACHINA_TOPIC_REVIEW_POSTS_NUMBER

Default: `10`

The number of posts displayed when posting a reply. The posts displayed are related to the considered forum topic.

2.3.4 Polls

MACHINA_POLL_MAX_OPTIONS_PER_POLL

Default: `30`

This setting can be used to configure the maximum number of options that can be defined when creating a poll.

MACHINA_POLL_MAX_OPTIONS_PER_USER

Default: `10`

This setting defines the maximum number of poll options that can be selected by users when voting. Note that this setting does not impact the users who vote in a poll but only the poll creator. The latest has to choose the number of poll options allowed per user, and this value cannot exceed the value of this setting.

2.3.5 Attachments

MACHINA_ATTACHMENT_FILE_UPLOAD_TO

Default: `'machina/attachments'`

The media subdirectory where forum attachments should be uploaded.

MACHINA_ATTACHMENT_CACHE_NAME

Default: `'machina_attachments'`

The name of the cache used to store temporary post attachments.

MACHINA_ATTACHMENT_MAX_FILES_PER_POST

Default: `15`

This setting can be used to configure the maximum number of attachments that can be associated to a forum post.

2.3.6 Member

MACHINA_PROFILE_AVATAR_UPLOAD_TO

Default: `'machina/avatar_images'`

The media subdirectory where forum member avatars should be uploaded.

MACHINA_PROFILE_AVATAR_WIDTH

Default: `150`

The width to use in order to resize forum profile avatars during upload. The image is not resized if this setting is set to `None`.

MACHINA_PROFILE_AVATAR_HEIGHT

Default: `250`

The height to use in order to resize forum profile avatars during upload. The image is not resized if this setting is set to `None`.

MACHINA_PROFILE_AVATAR_MIN_WIDTH

Default: `None`

The imposed avatar minimum width for forum member profiles. This setting affects avatars validation rules ; it should not be used jointly with the `MACHINA_PROFILE_AVATAR_WIDTH` and `MACHINA_PROFILE_AVATAR_HEIGHT` settings.

MACHINA_PROFILE_AVATAR_MIN_HEIGHT

Default: `None`

The imposed avatar minimum height for forum member profiles. This setting affects avatars validation rules ; it should not be used jointly with the `MACHINA_PROFILE_AVATAR_WIDTH` and `MACHINA_PROFILE_AVATAR_HEIGHT` settings.

MACHINA_PROFILE_AVATAR_MAX_WIDTH

Default: `None`

The imposed avatar maximum width for forum member profiles. This setting affects avatars validation rules ; it should not be used jointly with the `MACHINA_PROFILE_AVATAR_WIDTH` and `MACHINA_PROFILE_AVATAR_HEIGHT` settings.

MACHINA_PROFILE_AVATAR_MAX_HEIGHT

Default: `None`

The imposed avatar maximum height for forum member profiles. This setting affects avatars validation rules ; it should not be used jointly with the `MACHINA_PROFILE_AVATAR_WIDTH` and `MACHINA_PROFILE_AVATAR_HEIGHT` settings.

MACHINA_PROFILE_AVATAR_MAX_UPLOAD_SIZE

Default: `0`

The maximum avatar size for forum member profiles (the size must be expressed in bytes). A value of `0` means that there is no size limitation.

MACHINA_PROFILE_SIGNATURE_MAX_LENGTH

Default: `255`

The maximum number of characters that can be used in a member signature.

MACHINA_PROFILE_RECENT_POSTS_NUMBER

Default: `15`

The maximum number of recent posts that can be displayed in forum member profiles.

2.3.7 Permission

MACHINA_DEFAULT_AUTHENTICATED_USER_FORUM_PERMISSIONS

Default: `[]`

Django-machina relies on a permission system based on per-forum permissions. This allows you to define which permissions should be applied for each forum, for each user and for each group of users. However you might want to not have to deal with complex permissions and grant the same basic permissions to all the users and for all the forums you created. In that case, this setting can be used in order to define which permissions should be granted to all authenticated users. Note that the permissions specified in this list are granted only if the considered forum does not have any permission for the considered authenticated user. For example, the setting could be specified as follows:

```
MACHINA_DEFAULT_AUTHENTICATED_USER_FORUM_PERMISSIONS = [
    'can_see_forum',
    'can_read_forum',
    'can_start_new_topics',
    'can_reply_to_topics',
    'can_edit_own_posts',
    'can_post_without_approval',
    'can_create_polls',
    'can_vote_in_polls',
    'can_download_file',
]
```

For a full list of the available forum permissions, please refer to Forum permissions.

Note: Keep in mind that the permissions specified in the `MACHINA_DEFAULT_AUTHENTICATED_USER_FORUM_PERMISSIONS` list will be automatically granted for authenticated users if the targetted forum has no other permissions for these users. This behavior will apply if you create a new forum without a specific permission configuration ; so be careful with the permission code names you put in this setting.

2.4 Glossary

This is a comprehensive list of the terms used when discussing the functionalities of *django-machina*.

Attachment An attachment is file associated with a forum message that other forum users may see in order to download it.

Forum A forum is a container for messages. It is caracterized by a name and can be part of a tree of other forums. That way a forum may have a parent forum and multiples sub-forums. A forum is typed and can correspond to a **default forum**, a **category** or a **forum link**. A **default forum** contains mesages and can have sub-forums. A **category** can only contains default forums. A **forum link** redirects to a specified link and cannot have sub-forums.

Forum permission Forum permissions define what actions a user (anonymous or not) can do or not in a specific forum (eg. answer to forum topics).

Post A post is a message embedded into a conversation that was submitted by a forum user. A post usually consists of a title and a text, but can also contain attachments.

Topic A forum topic represents a conversation between forum users. It contains messages (or "posts") that were submitted by the forum users. A topic generally refers to the name of the conversation and the first message (or "post") embedded into it. A forum topic may contain additional contents like polls. A forum topics can be typed and can correspond to a **normal topic**, a **sticky topic** or an **announcement**. A **normal topic** is a regular

conversation that will slide down the forum if no other posts are created into it and get bumped to the top of the forum otherwise. A **sticky topic** is a topic that is stuck at the top of the first page of a forum. An **announcement** is a topic that is stuck at the top of every page of a forum.

2.5 Forum permissions

Django-machina comes with its own permission system, allowing you to define exactly what users or groups can or can not do with the forums you created.

Permissions can be granted to users (anonymous user or registered users) and to groups. Some permissions can be granted globally: in this case, the permissions apply to all forums.

2.5.1 Built-in permissions

Permission	Is global	Definition
Forums		
`can_see_forum`	Yes	Defines whether the target can see a forum (eg. in the list of forums)
`can_read_forum`	Yes	Defines whether the target can read the content of a forum
Topics and posts		
`can_start_new_topics`	Yes	Defines whether the target can start a new topic
`can_start_new_topics`	Yes	Defines whether the target can start a new topic
`can_reply_to_topics`	Yes	Defines whether the target can reply to topics
`can_post_announcements`	Yes	Defines whether the target can create announces
`can_post_stickies`	Yes	Defines whether the target can create sticky posts
`can_delete_own_posts`	Yes	Defines whether the target can remove its own posts
`can_edit_own_posts`	Yes	Defines whether the target can edit its own posts
`can_post_without_approval`	Yes	Defines whether the target can create topics or posts without moderator approval
Polls		
`can_create_polls`	Yes	Defines whether the target can create polls
`can_vote_in_polls`	Yes	Defines whether the target can vote in polls
Attachments		
`can_attach_file`	Yes	Defines whether the target can attach files to forum posts
`can_download_file`	Yes	Defines whether the target can download the files attached to forum posts
Moderation		
`can_lock_topics`	No	**Moderation permission:** defines whether the target can lock a forum topic
`can_move_topics`	No	**Moderation permission:** defines whether the target can move topics to another forum
`can_edit_posts`	No	**Moderation permission:** defines whether the target can edit forum posts that he did not write
`can_delete_posts`	No	**Moderation permission:** defines whether the target can delete forum posts that he did not write
`can_approve_posts`	No	**Moderation permission:** defines whether the target can approve unapproved posts
`can_reply_to_locked_topic`	No	**Moderation permission:** defines whether the target can add posts in locked topics

2.5.2 Defining forum permissions

Django-machina allows you to precisely define which permissions should be granted for each forum, for each user and for each group of users. The permissions can be granted from the administration panel. Just go to the 'Forums' section of the administration panel. In this section you can update forum instances and their related permissions.

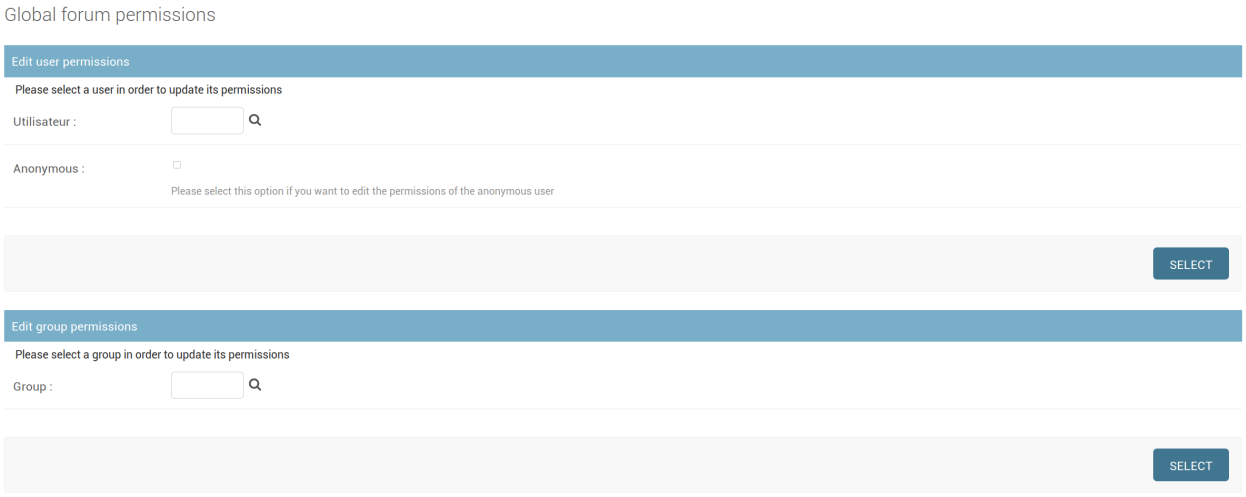
Note: Defining precise permissions on each forum can be overwhelming if you just want to set up single forums with a basic set of permissions. In that case you can use the `MACHINA_DEFAULT_AUTHENTICATED_USER_FORUM_PERMISSIONS` setting to define which permissions should be granted to all authenticated users for all forums (please refer to Settings).

As previously stated, the forum permissions can be applied either to a specific forum or globally to all forums:

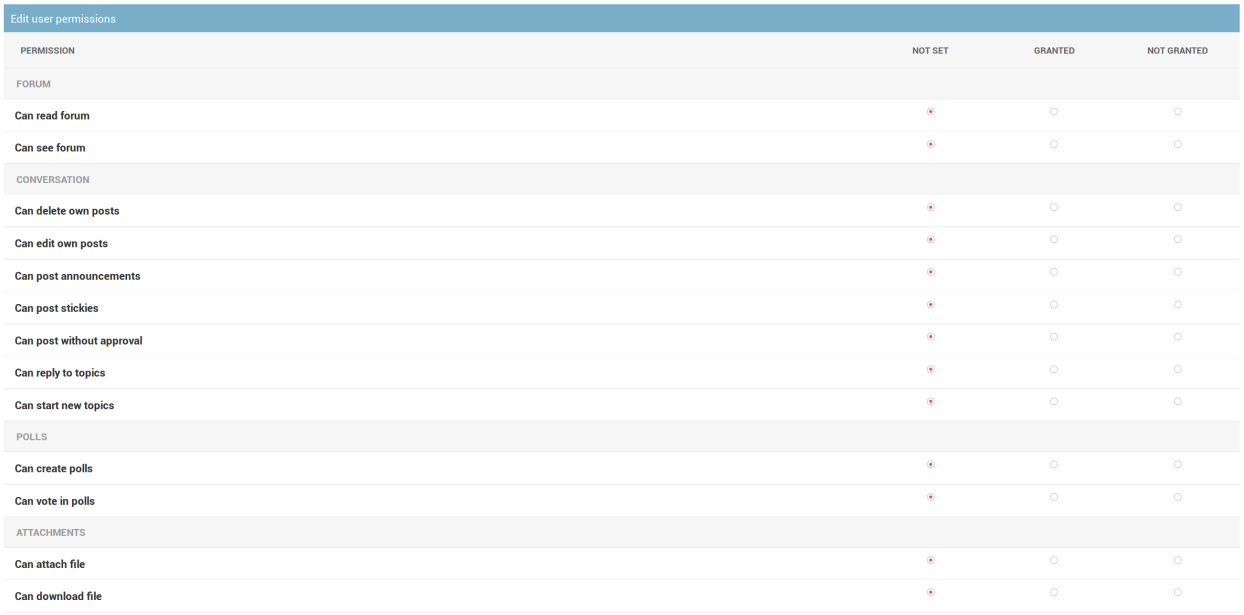
- in order to edit global forum permissions, go to the list of forum instances in the administration panel and click on "Global forum permissions"
- in order to edit specific forum permissions, select a forum in the list of forum instances in the administration panel. Then click on "Forum permissions"

Note that global permissions have a lower priority than permissions that are associated with a specific forum. For example, a forum will be hidden if it is tied with a permission defining that it should not be accessible for a group of user, even if this forum can be accessed according to the global permissions applying to all forums for this group of users.

The admin pages mentioned above ("Global forum permissions" or "Forum permissions" for specific forums) allow you to select the user or group for which you want to set permissions. You have to select a specific user, the anonymous user or a specific group in order to set its permissions.

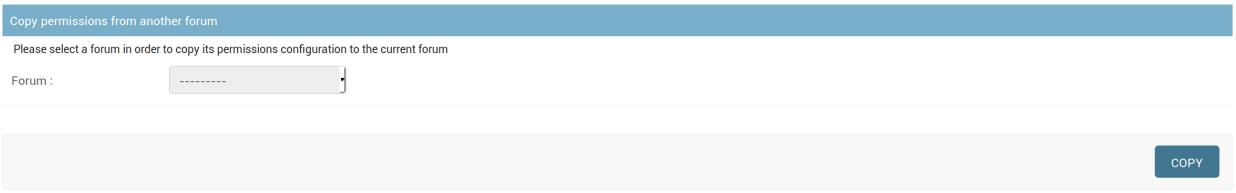


Once you have selected a user or group, you access a page where you can set its permissions for the considered forum (or for all the forums in case of global permissions). The form allows you to define the state of each permission for the considered user or group. Each permission can be either **not set**, **granted** or **not granted**.



2.5.3 Copying forum permissions

If you are on the permissions page of a specific forum, you can choose to copy the permissions configuration of another forum in order to apply it to the current forum. This allows you to easily apply a set of permission to many forums.



2.6 Customization

Django-machina was built with customization in mind. The module provides useful tools to make your forum compatible with your own business logic.

2.6.1 Settings

As most Django applications do, *django-machina* allows you to customize your forum application with a set of settings (please refer to Settings). *Django-machina*'s settings cover many aspects of your forum: markup language, pagination, images, default permissions, etc.

2.6.2 Templates and static files

If you wish to personalize the look and feel of your forum you can take advantage of the Django's template loading system. Thus you can easily override forum layouts and styles if Django is configured to look in your project first for templates before using the *django-machina*'s templates.

For example, you can easily override *django-machina*'s templates by configuring the `TEMPLATE_DIRS` setting as follows:

```
import os

TEMPLATE_DIRS = (
  os.path.join(PROJECT_PATH, 'src/vanilla_project/templates'),
  MACHINA_MAIN_TEMPLATE_DIR,
)
```

2.6.3 Advanced customization mechanisms

Django-machina relies on a dynamic class loading system that allows to override or extend its Python classes: class-based views, forms, models, etc. This gives you the power to adapt your forum to your own business logic.

In order to benefit from this dynamic class loading system, you will need to override a *django-machina* application. Please head over to the following topics in order to achieve this:

Overriding an application

Django-machina relies on a dynamic class-loading system that allows you to override or extend many aspects of its applications. The *django-machina* applications are listed below:

Application name	Definition
forum	This application provides the ability to browse a tree of forums
forum_conversation	This application handles all the conversations that can happen in forums
forum_feeds	This application allows to get forum topics as RSS feeds
forum_member	This application provides functionalities to forum members
forum_moderation	This application provides moderation tools to moderators
forum_permission	This application provides the proper tools to allow permission checks on forums
forum_search	This application allows to search within forums
forum_tracking	This application allows to determine which forums or topics have been read by a given user

Note: Overriding these applications is not a trivial task. Most of the time you will need to dig into the source code of *django-machina* in order to discover how things were implemented. This will allow you to find exactly which method should be rewritten in order to achieve the task at hand.

Duplicate the application

Let's say we want to override the `machina.apps.forum_conversation` application.

Create a Python package with the same application label The first thing to do is to create a Python package with the same application label as the app you want to override. This package can live under an `apps` Python package that acts as a root folder for your overridden applications, as shown below:

```
$ mkdir -p apps/forum_conversation
$ touch apps/__init__.py
$ touch apps/forum_conversation/__init__.py
```

Import the application models if needed All *django-machina*'s applications do not necessarily contain models. So this step may be skipped depending on the application you want to override. In the other case, it is necessary to reference the models of the overridden application by creating a `models.py` file in your package:

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals

# Custom models should be declared before importing
# django-machina models

from machina.apps.forum_conversation.models import *  # noqa
```

Your overridden application may need to add new models or modify *django-machina*'s own models. As stated in this snippet, custom models must be declared **before** the import of the *django-machina*'s models. This means that you can override a *django-machina* model in order to change the way it behaves if you want. Please refer to Overriding application models to get detailed instructions on how to override *django-machina*'s models.

Only importing *django-machina*'s models is not enough. You have to ensure the models migrations can be used by your Django project. You have two possibilities to do so:

- you can copy the content of the `migrations` folder from the application you want to override to your own local application
- you can configure the `MIGRATION_MODULES` setting to reference the original migrations of the application you want to override

```
MIGRATION_MODULES = {
  'forum_conversation': 'machina.apps.forum_conversation.migrations',
}
```

Note: The second possibility should only be used if you are sure you will not define new models or overridden models into your local application

Import the application admin classes if needed As previously stated, this step can be skipped if the application you want to override does not contain models. In the other case you will want to create an `admin.py` file in your package in order to reference the admin classes of the overridden application:

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals
from machina.apps.forum_conversation.admin import *  # noqa
```

Use the application AppConfig Most of *django-machina*'s applications define sublclasses of Django's `AppConfig` which can perform initialization operations. *Django-machina* `AppConfig` instances are defined inside sub-modules called `registry_config`. You need to make sure the `AppConfig` subclass of the application you want to override is properly loaded. So your application's `__init__.py` should include the default app config to use:

```
default_app_config = 'machina.apps.forum_conversation.registry_config.ConversationRegistryConfig'
```

Add the local application to your INSTALLED_APPS

Finally you have to tell Django to use your overridden application instead of the *django-machina*'s original application. You can do this by adding your application as a second argument to the `get_apps` function in your Django settings:

```
from machina import get_apps as get_machina_apps

INSTALLED_APS = [
  # ...
] + get_machina_apps(['yourproject.apps.forum_conversation', ])
```

The list you pass to the `get_apps` function must contain overridden applications.

Customization underlying mechanisms

Django-machina relies on a dynamic class-loading system that allows you to override or extend many aspects of its applications. The underlying mechanisms are directly inspired from the class loading system provided by the django-oscar e-commerce framework.

If you look through *django-machina*'s codebase, you'll find that most of the classes or functions are imported using this kind of statement:

```
from machina.core.loading import get_class
PostForm = get_class('forum_conversation.forms', 'PostForm')
```

The `get_class` function is provided by the `machina.core.loading` module. It is used instead of standard import statements such as `from machina.forum_conversation.forms import PostForm`.

The `get_class` function imports a single class from a specified module. It takes two arguments: the first one is the label of the module from which you want to import your class (eg. `forum_conversations.forms`) ; the second-one is the name of the class to import. The `get_class` function works as follow:

- it will look through your Machina overridden applications in order to find an application that matches the application name included in the module label
- it will try to load the class from the specified module if it exists
- if the specified module is not present in the overriden application or if the class cannot be retrieved from the custom module, the class will be imported from the default Machina application

---

Note: The `get_class` function can only import customized classes from applications that have been properly overridden. Please head over to Overriding an application for more details on how to override a *django-machina* application.

So the `get_class` function allows you to define local versions of Machina classes in order to customize your forum behaviors. Most of the time you will create a subclass of a specific class in order to customize the way it behaves. For example you could extend the `forum_conversation.views.TopicView` in order to add some data to the context:

```
from __future__ import unicode_literals

from machina.apps.forum_conversation.views import TopicView as BaseTopicView

class TopicView(BaseTopicView):
  def get_context_data(self, **kwargs):
    context = super(TopicView, self).get_context_data(**kwargs)
    # Some additional data can be added to the context here
```

```
    context['foo'] = 'bar'
    return context
```

If this view is part of an overridden application, *django-machina* will use it instead of the default `TopicView`.

So this dynamic class-loading system allows to make changes to the *django-machina*'s core functionalities by altering only the classes whose behavior must be updated to achieve the task at hand.

2.6.4 Recipes

Here is a list of simple guides demonstrating how to solve common customization problems when using *django-machina*:

Overriding application models

Django-machina allows you to override its models. This can be useful if you want to add new methods or new fields to existing *django-machina* models.

To illustrate this functionality, we will add an `icon` field to the `Topic` model (which is part of the `forum_conversation` app) in order to allow users select an icon for the topics they create.

Prerequisite

Please ensure that you have correctly followed the instructions described in Overriding an application before trying to override *django-machina* models. If so, you should have created a Python package with the same application label as the app you want to override. This new application should be defined in your `INSTALLED_APPS` setting.

Most importantly, you should've created a `models.py` file inside your package in order to reference the models of the overriden application:

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals

# Custom models should be declared before importing
# django-machina models

from machina.apps.forum_conversation.models import *  # noqa
```

Finally you should have copied the content of the `migration` folder from the application you want to override into your own local application.

Defining a new custom model

In order to define a new version of an existing *django-machina* model you have to define a new class that subclasses the abstract model class of the model you want to override. The new model you define must have the exact same name as the model you are trying to override.

For example, in order to define a custom version of the `Topic` model it is necessary to subclass the `machina.apps.forum_conversation.abstract_models.AbstractTopic` abstract model:

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals
```

```
from machina.apps.forum_conversation.abstract_models import AbstractTopic

# Custom models should be declared before importing
# django-machina models

class Topic(AbstractTopic):
    icon = models.ImageField(verbose_name="Icon", upload_to="forum/topic_icons")

from machina.apps.forum_conversation.models import *  # noqa
```

Note: You need to ensure that the import of *django-machina*'s models is always done at the bottom of your `models.py` file. This is very important in the event that you define overridden models because it will ensure that your overriden models will be loaded by Django instead of the original versions provided by *django-machina*.

Creating migrations

As stated previously, you should've copied the content of the `migration` folder from the application you want to override into your own local application. Then you just have to create a new migration related to the changes you made to the overriden models:

```
$ django-admin makemigrations forum_conversations
```

Using another markup language for forum posts

Django-machina uses Markdown as the default syntax for forum messages, which is provided by the use of the django-markdown module. But you can easily change this in your settings. We will see how to do this.

It should be noted that *django-machina* relies on specific model fields to store forum messages. These fields contribute two columns to the model where they are used: the first one is used to store any content written by using a markup language (eg. BBCode or Markdown) and the second one keeps the rendered content obtained by converting the initial content to HTML. Thus forum messages are stored in two versions: plain and HTML.

Example: using django-ckeditor

Let's use django-ckeditor instead of django-markdown in order to benefit from a powerful wysiwyg editor.

The first thing to do is to replace `django_markdown` by `ckeditor` in our `INSTALLED_APPS` setting:

```
from machina import get_apps as get_machina_apps

INSTALLED_APS = [
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.sites',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'django.contrib.admin',

  # Machina related apps:
  'mptt',
  'haystack',
```

```
  'widget_tweaks',
  'ckeditor',
] + get_machina_apps()
```

Then we must set the `MACHINA_MARKUP_LANGUAGE` and `MACHINA_MARKUP_WIDGET` settings in order to tell *django-machina* the widget to use when displaying forms:

```
MACHINA_MARKUP_LANGUAGE = None
MACHINA_MARKUP_WIDGET = 'ckeditor.widgets.CKEditorWidget'
```

When using a wysiwyg editor such as CKEditor we do not use a specific markup language because we directly get the content in HTML. This is why the `MACHINA_MARKUP_LANGUAGE` setting is set to `None`. The `MACHINA_MARKUP_WIDGET` indicates the Python dotted path to the CKEditor form widget.

The last thing to do is to ensure that you use the required assets in your templates. Basically, you have to ensure that the media property is used in your form templates (this is the case if you have not modified the default topic/post templates):

```
{% block css %}
  {{ block.super }}
  {{ post_form.media.css }}
{% endblock css %}

{% block js %}
  {{ block.super }}
  {{ post_form.media.js }}
{% endblock js %}
```

2.7 Machina's apps reference

Django-machina is organized into several applications. Each application provides specific forum features such as forum display, permission checks, etc.

2.7.1 Forum

The `forum` application provides the ability to browse a tree of forums. It is based on a single `AbstractForum` abstract model.

Abstract models

class `machina.apps.forum.abstract_models.`**`AbstractForum`**(**args*, ***kwargs*)

Bases: `mptt.models.MPTTModel`, `machina.models.abstract_models.ActiveModel`, `machina.models.abstract_models.DatedModel`

The main forum model. The tree hierarchy of forums and categories is managed by the MPTTModel which is part of django-mptt.

`is_category`
Returns True if the forum is a category.

`is_forum`
Returns True if the forum is a a default forum.

`is_link`
Returns True if the forum is a link.

margin_level
Used in templates or menus to create an easy-to-see left margin to contrast a forum from their parents.

Views

class machina.apps.forum.views.**ForumView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.list.ListView

Displays a forums and its topics. If applicable, its sub-forums can also be displayed.

get_forum()
Returns the forum to consider.

class machina.apps.forum.views.**IndexView**(***kwargs*)
Bases: django.views.generic.list.ListView

Displays the top-level forums.

2.7.2 Forum conversation

The forum_conversation application handles all the conversations that can happen in forums. It provides some of the main features of a forum application: posting messages, writing answers, voting in polls, ...

Sub applications

Forum attachments

The forum_attachments application handles all the attachments that can be associated with forum posts.

Abstract models
class machina.apps.forum_conversation.forum_attachments.abstract_models.**AbstractAttachment**(**a*
****

Bases: django.db.models.base.Model

Represents a post attachment. An attachment is always linked to a post.

Cache
class machina.apps.forum_conversation.forum_attachments.cache.**AttachmentCache**
Bases: object

The attachments cache. This one should be used with a FileBasedCache backend. But this can be overriden. The attachments cache acts as a wrapper and ensure that the states (name, size, content type, charset and content) of all files from any request.FILES dict are saved inside the considered backend when calling the 'set' method. Conversely, the 'get' method will populate a dictionary of InMemoryUploadedFile instances or TemporaryUploadedFile instancesby using these states.

get(*key*)
Regenerates a MultiValueDict instance containing the files related to all file states stored for the given key.

set(*key*, *files*)
Stores the state of each file embedded in the request.FILES MultiValueDict instance. This instance is assumed to be passed as the 'files' argument. Each state stored in the cache is a dictionary containing the following values:

name The name of the uploaded file.

size The size of the uploaded file.

content_type The content type of the uploaded file.

content_length The content length of the uploaded file.

charset The charset of the uploaded file.

content The content of the uploaded file.

Views

class machina.apps.forum_conversation.forum_attachments.views.**AttachmentView**(***kwargs*)

Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.detail.DetailView

Allows to retrieve a forum attachment.

model

alias of Attachment

Forum polls

The forum_polls application handles all the polls that can be created in forum topics. It provides forms and tools for creating polls and for voting in forum polls.

Abstract models

class machina.apps.forum_conversation.forum_polls.abstract_models.**AbstractTopicPoll**(**args*, ***kwargs*)

Bases: machina.models.abstract_models.DatedModel

Represents a poll embedded in a forum topic.

votes

Returns all the votes related to this topic poll.

class machina.apps.forum_conversation.forum_polls.abstract_models.**AbstractTopicPollOption**(**ar* ***k*

Bases: django.db.models.base.Model

Represents a poll option.

class machina.apps.forum_conversation.forum_polls.abstract_models.**AbstractTopicPollVote**(**args*, ***kwa*

Bases: django.db.models.base.Model

Represents a poll vote.

Views

class machina.apps.forum_conversation.forum_polls.views.**TopicPollVoteView**(***kwargs*)

Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Allows to vote in polls.

model

alias of TopicPoll

Abstract models

class machina.apps.forum_conversation.abstract_models.**AbstractPost**(**args*, ***kwargs*)

Bases: machina.models.abstract_models.DatedModel

Represents a forum post. A forum post is always linked to a topic.

is_topic_head
Returns True if the post is the first post of the topic.

is_topic_tail
Returns True if the post is the last post of the topic.

position
Returns an integer corresponding to the position of the post in the topic.

class machina.apps.forum_conversation.abstract_models.**AbstractTopic**(**args*, ***kwargs*)

Bases: machina.models.abstract_models.DatedModel

Represents a forum topic.

first_post
Try to fetch the first post associated with the current topic and caches it to lighten the next request.

is_announce
Returns True if the topic is an announce.

is_locked
Returns True if the topic is locked.

is_sticky
Returns True if the topic is a sticky topic.

is_topic
Returns True if the topic is a default topic.

last_post
Try to fetch the last post associated with the current topic and caches it to lighten the next request.

update_trackers()
Updates the posts count, the update date and the link toward the last post associated with the current topic.

Views

class machina.apps.forum_conversation.views.**BasePostFormView**(***kwargs*)

Bases: django.views.generic.edit.FormView

A base view for handling post forms.

form_invalid(*post_form*, *attachment_formset*, ***kwargs*)
Called if one of the forms is invalid. Re-renders the context data with the data-filled forms and errors.

form_valid(*post_form*, *attachment_formset*, ***kwargs*)
Called if all forms are valid. Creates a Post instance along with associated attachments if required and then redirects to a success page.

get_attachment_formset(*formset_class*)
Returns an instance of the attachment formset to be used in the view.

get_attachment_formset_class()
Returns the attachment formset class to use for instantiating the attachment formset.

get_attachment_formset_kwargs()
Returns the keyword arguments for instantiating the attachment formset.

get_attachments_cache_key(*request*)
Returns the key used to store attachment files states into the file based cache.

get_forum()
Returns the considered forum.

get_post()
Returns the considered post if applicable.

get_post_form(*form_class*)
Returns an instance of the post form to be used in the view.

get_post_form_class()
Returns the post form class to use for instantiating the form.

get_post_form_kwargs()
Returns the keyword arguments for instantiating the post form.

get_topic()
Returns the considered topic if applicable.

init_attachment_cache()
Initializes the attachment cache for the current view.

class machina.apps.forum_conversation.views.**BaseTopicFormView**(***kwargs*)
Bases: *machina.apps.forum_conversation.views.BasePostFormView*

A base view for handling topic forms.

get_poll_option_formset(*formset_class*)
Returns an instance of the poll option formset to be used in the view.

get_poll_option_formset_class()
Returns the poll option formset class to use for instantiating the poll option formset.

get_poll_option_formset_kwargs()
Returns the keyword arguments for instantiating the poll option formset.

class machina.apps.forum_conversation.views.**PostCreateView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, *machina.apps.forum_conversation.views.PostFormView*

Allows users to create forum posts.

model
alias of Post

class machina.apps.forum_conversation.views.**PostDeleteView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Allows users to delete forum topics.

delete(*request*, **args*, ***kwargs*)
Calls the delete() method on the fetched object and then redirects to the success URL. This is a workaround for versions of Django prior 1.6 where the get_success_url() method was called after the delete() method.

model
alias of Post

class machina.apps.forum_conversation.views.**PostFormView**(***kwargs*)
Bases: django.views.generic.detail.SingleObjectMixin, *machina.apps.forum_conversation.views.BasePostFormView*

A base view for manipulating post forms.

class machina.apps.forum_conversation.views.**PostUpdateView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, *machina.apps.forum_conversation.views.PostFormView*

Allows users to update forum topics.

model
alias of Post

class machina.apps.forum_conversation.views.**TopicCreateView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, *machina.apps.forum_conversation.views.TopicFormView*

Allows users to create forum topics.

model
alias of Topic

class machina.apps.forum_conversation.views.**TopicFormView**(***kwargs*)
Bases: django.views.generic.detail.SingleObjectMixin, *machina.apps.forum_conversation.views.BaseTopicFormView*

A base view for manipulating topic forms.

class machina.apps.forum_conversation.views.**TopicUpdateView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, *machina.apps.forum_conversation.views.TopicFormView*

Allows users to update forum topics.

get_post()
Returns the considered post if applicable.

model
alias of Topic

class machina.apps.forum_conversation.views.**TopicView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.list.ListView

Displays a forum topic.

get_topic()
Returns the topic to consider.

2.7.3 Forum feeds

The forum_feeds application allows to get forum topics as RSS feeds.

Feeds

class machina.apps.forum_feeds.feeds.**LastTopicsFeed**
Bases: django.contrib.syndication.views.Feed

Provides feed items for the latest forum topics.

2.7.4 Forum member

The `forum_member` application provides functionalities to forum members and defines forum profile abstract models.

Abstract models

class `machina.apps.forum_member.abstract_models.`**`AbstractForumProfile`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Represents the profile associated with each forum user.

Views

class `machina.apps.forum_member.views.`**`ForumProfileDetailView`**(***kwargs*)

Bases: `django.views.generic.detail.DetailView`

Shows a user's forum profile.

class `machina.apps.forum_member.views.`**`ForumProfileUpdateView`**(***kwargs*)

Bases: `django.views.generic.edit.UpdateView`

Allows the current user to update its forum profile.

class `machina.apps.forum_member.views.`**`UserTopicsView`**(***kwargs*)

Bases: `django.views.generic.list.ListView`

Provides a list of all the topics in which the current user has posted messages.

2.7.5 Forum moderation

The `forum_moderation` application provides moderation views allowing moderators to move, close or delete forum topics or posts. It also provides access to the moderation queue used to approve or reject posts awaiting approval.

Views

class `machina.apps.forum_moderation.views.`**`PostApproveView`**(***kwargs*)

Bases: `machina.apps.forum_permission.viewmixins.PermissionRequiredMixin`, `django.views.generic.detail.SingleObjectTemplateResponseMixin`, `django.views.generic.detail.BaseDetailView`

A view providing the ability to approve queued forum posts.

`approve`(*request*, **args*, ***kwargs*)

Approves the considered post and retirects the user to the success URL.

`model`

alias of `Post`

class `machina.apps.forum_moderation.views.`**`PostDisapproveView`**(***kwargs*)

Bases: `machina.apps.forum_permission.viewmixins.PermissionRequiredMixin`, `django.views.generic.detail.SingleObjectTemplateResponseMixin`, `django.views.generic.detail.BaseDetailView`

A view providing the ability to disapprove queued forum posts.

disapprove(*request*, **args*, ***kwargs*)
Disapproves the considered post and retirects the user to the success URL.

model
alias of `Post`

class machina.apps.forum_moderation.views.**TopicDeleteView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

A view providing the ability to delete forum topics.

delete(*request*, **args*, ***kwargs*)
Deletes the considered topic and retirects the user to the success URL. Calls the delete() method on the fetched object and then redirects to the success URL. This is a workaround for versions of Django prior 1.6 where the get_success_url() method was called after the delete() method.

model
alias of `Topic`

class machina.apps.forum_moderation.views.**TopicLockView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.detail.SingleObjectTemplateResponseMixin, django.views.generic.detail.BaseDetailView

A view providing the ability to lock forum topics.

lock(*request*, **args*, ***kwargs*)
Locks the considered topic and retirects the user to the success URL.

model
alias of `Topic`

class machina.apps.forum_moderation.views.**TopicMoveView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.detail.SingleObjectTemplateResponseMixin, django.views.generic.edit.FormMixin, django.views.generic.detail.SingleObjectMixin, django.views.generic.edit.ProcessFormView

A view providing the ability to move forum topics.

model
alias of `Topic`

class machina.apps.forum_moderation.views.**TopicUnlockView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.detail.SingleObjectTemplateResponseMixin, django.views.generic.detail.BaseDetailView

A view providing the ability to unlock forum topics.

model
alias of `Topic`

unlock(*request*, **args*, ***kwargs*)
Unlocks the considered topic and retirects the user to the success URL.

class machina.apps.forum_moderation.views.**TopicUpdateTypeBaseView**(***kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.detail.SingleObjectTemplateResponseMixin, django.views.generic.detail.BaseDetailView

A view providing the ability to change the type of forum topics: normal, sticky topic or announce.

model
alias of Topic

update_type(*request*, **args*, ***kwargs*)
Updates the type of the considered topic and retirects the user to the success URL.

2.7.6 Forum permission

The forum_permission application provides the proper tools to allow permission checks on forums. It defines permission abstract models and provides

Abstract models

class machina.apps.forum_permission.abstract_models.**AbstractForumPermission**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Represents a single forum permission.

class machina.apps.forum_permission.abstract_models.**AbstractGroupForumPermission**(**args*, ***kwargs*)
Bases: *machina.apps.forum_permission.abstract_models.BaseAuthForumPermission*

Represents a per-group forum object permission.

class machina.apps.forum_permission.abstract_models.**AbstractUserForumPermission**(**args*, ***kwargs*)
Bases: *machina.apps.forum_permission.abstract_models.BaseAuthForumPermission*

Represents a per-user forum object permission.

class machina.apps.forum_permission.abstract_models.**BaseAuthForumPermission**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Represents a per-auth-component forum object permission.

Checker

class machina.apps.forum_permission.checker.**ForumPermissionChecker**(*user*)
Bases: object

The ForumPermissionChecker allows to check forum permissions on Forum instances.

get_perms(*forum*)
Returns the list of permission codenames of all permissions for the given forum.

has_perm(*perm*, *forum*)
Checks if the considered user has given permission for the passed forum.

Handler

class machina.apps.forum_permission.handler.**PermissionHandler**
Bases: object

The PermissionHandler allows to filter lists of forums and to perform permission verifications on forums. It uses the ForumPermissionChecker class to perform these verifications.

can_access_moderation_queue (*user*)
Returns True if the passed user can access the moderation queue. The latest allows the moderator to approve posts.

can_add_announcements (*forum*, *user*)
Given a forum, checks whether the user can append announcements to it.

can_add_post (*topic*, *user*)
Given a topic, checks whether the user can append posts to it.

can_add_stickies (*forum*, *user*)
Given a forum, checks whether the user can append stickies to it.

can_add_topic (*forum*, *user*)
Given a forum, checks whether the user can append topics to it.

can_approve_posts (*forum*, *user*)
Given a forum, checks whether the user can approve its posts.

can_attach_files (*forum*, *user*)
Given a forum, checks whether the user can add attachments to posts.

can_create_polls (*forum*, *user*)
Given a forum, checks whether the user can add a topic with an embedded poll.

can_delete_post (*post*, *user*)
Given a forum post, checks whether the user can delete the latter.

can_delete_topics (*forum*, *user*)
Given a forum, checks whether the user can delete its topics. Note: the `can_delete_posts` permission is used here because a user who can delete all the posts of a topic is also able to delete the topic itself.

can_download_files (*forum*, *user*)
Given a forum, checks whether the user can download files attached to posts.

can_edit_post (*post*, *user*)
Given a forum post, checks whether the user can edit the latter.

can_lock_topics (*forum*, *user*)
Given a forum, checks whether the user can lock its topics.

can_move_topics (*forum*, *user*)
Given a forum, checks whether the user can move its topics to another forum.

can_post_without_approval (*forum*, *user*)
Given a forum, checks whether the user can add a posts and topics without approval.

can_read_forum (*forum*, *user*)
Given a forum, checks whether the user can read its content.

can_update_topics_to_announces (*forum*, *user*)
Given a forum, checks whether the user can change its topic types to announces.

can_update_topics_to_normal_topics (*forum*, *user*)
Given a forum, checks whether the user can change its topic types to normal topics.

can_update_topics_to_sticky_topics (*forum*, *user*)
Given a forum, checks whether the user can change its topic types to sticky topics.

can_vote_in_poll (*poll*, *user*)
Given a poll, checks whether the user can answer to it.

forum_list_filter(*qs*, *user*)
Filters the given queryset in order to return a list of forums that can be seen or read by the specified user (at least).

get_forum_last_post(*forum*, *user*)
Given a forum, fetch the last post that can be read by the passed user.

get_moderation_queue_forums(*user*)
Returns the list of forums whose posts can be approved by the considered user.

get_target_forums_for_moved_topics(*user*)
Returns a list of forums in which the considered user can add topics that have been moved from another forum.

2.7.7 Forum search

The `forum_search` application allows to search within forums.

Search indexes

class `machina.apps.forum_search.search_indexes.`**PostIndex**
Bases: `haystack.indexes.SearchIndex`, `haystack.constants.Indexable`

Defines the data stored in the Post indexes.

Views

class `machina.apps.forum_search.views.`**FacetedSearchView**(**args*, ***kwargs*)
Bases: `haystack.views.FacetedSearchView`

Allows to search within forums

2.7.8 Forum tracking

The `forum_tracking` application allows to determine which forums or topics have been read by a given user. It provides mechanisms allowing users to mark forums or topics as read and to browse unread forums or topics.

Handler

class `machina.apps.forum_tracking.handler.`**TrackingHandler**(*request=None*)
Bases: `object`

The TrackingHandler allows to filter list of forums and list of topics in order to get only the forums which contain unread topics or the unread topics.

get_unread_forums(*forums*, *user*)
Returns a list of unread forums for the given user from a given set of forums.

get_unread_topics(*topics*, *user*)
Returns a list of unread topics for the given user from a given set of topics.

mark_forums_read(*forums*, *user*)
Marks a list of forums as read.

mark_topic_read(*topic*, *user*)
Marks a topic as read.

Views

class machina.apps.forum_tracking.views.**MarkForumsReadView**(**kwargs*)
Bases: django.views.generic.base.View

Marks a set of forums as read.

class machina.apps.forum_tracking.views.**MarkTopicsReadView**(**kwargs*)
Bases: machina.apps.forum_permission.viewmixins.PermissionRequiredMixin, django.views.generic.base.View

Marks a set of topics as read.

get_controlled_object()
Return the considered forum in order to allow permission checks.

class machina.apps.forum_tracking.views.**UnreadTopicsView**(**kwargs*)
Bases: django.views.generic.list.ListView

Displays unread topics for the current user.

2.8 Contributing to django-machina

Here are some simple rules to help you contribute to *django-machina*. You can contribute in many ways!

2.8.1 Contributing code

The preferred way to contribute to *django-machina* is to submit pull requests to the project's Github repository. Here are some general tips regarding pull requests.

Warning: Keep in mind that you should propose new features on the project's issue tracker before starting working on your ideas! Remember that the central aim of *django-machina* is to provide a solid core of a forum project - without much of extra functionality included!

Development environment

You should first fork the django-machina's repository. Then you can get a working copy of the project using the following commands:

```
$ git clone git@github.com:<username>/django-machina.git
$ cd django-machina && mkvirtualenv machina
(machina) $ make install
```

Coding style

Please make sure that your code is compliant with the PEP8 style guide. You can ignore the "Maximum Line Length" requirement but you should still pay attention to the length of your lines. Remember that your code will be checked using flake8. You can use the *django-machina*'s tox configuration to perform this validation:

```
$ tox -e lint
```

Tests

You should not submit pull requests without providing tests. *Django-machina* uses pytest as a test runner but also as a syntax for tests instead of unittest. So you should write your tests using pytest instead of unittest and you should not use the built-in `django.test.TestCase`.

You can run the whole test suite using the following command:

```
$ py.test
```

Code coverage should not decrease with pull request! You can easily get the code coverage of the project using the following command:

```
$ make coverage
```

2.8.2 Contributing translations

The translation work on *django-machina* is done using Transifex. Don't hesitate to apply for a lanaguage if you want to improve the internationalization of the project.

2.8.3 Using the issue tracker

You should use the project's issue tracker if you've found a bug or if you want to propose a new feature. Don't forget to include as many details as possible in your tickets (eg. tracebacks if this is appropriate).

2.9 Indices and tables

- genindex
- modindex
- search

m

A

B

C

D

F

G

P

S

T

U

V