**Syntax highlighting with Django and Markdown**

by Martin Fitzpatrick — get latest articles [straight to your Inbox](http://mfitzp.us12.list-manage.com/subscribe/post?u=1e360ecc7f9998fd0be3ddb60&id=e16530b855).

Syntax highlighting is more than just eye candy. It can turn a block of impenetrable code into a simple grok (although it has it’s limits). To boost the usability of our code guides on this site we wanted to implement nice clear syntax highlighting of our hacks. Code samples are already marked up with markdown syntax so an extension to this is the obvious choice.

Here we document the process of implementing syntax highlighting on markdown marked-up(!) code blocks.



The resulting markup is heavily dependent on whether the language is detected. You can tell the parser what language you are using with a few simple lines at the top of your code:

Start your code with a shebang with path, and the language will be derived from that, with line numbers

#!/usr/bin/python

Use a shebang without the path and the shebang line will be removed:

#!python

Start the first line with three colons (:::) and the following text will be used to identify the language. The line is removed and no numbers are used:

:::python

If you use none of these systems the codehilite plugin will attempt to guess the language, but it might not get it right (resulting in some inconsistent code highlighting).

## Method

If you haven’t already, install python-markdown. You need this to mark up your code, but this is not a tutorial for python-markdown use.

Download and install [pygments](http://pygments.org/).

Add both to INSTALLED\_APPS in your settings.py file in Django.

INSTALLED\_APPS **=** (

**...**

'django.contrib.comments',

'mptt',

'comments',

**...**

)

In your template file you can use markdown by including the **markup** tags :

***{%*** **load** markup ***%}***

***{{*** object.content**|markdown** ***}}***

Markdown supports extensions, which you can supply to the template tag filter. Django-markdown already comes with support for pygments included, as long as pygments is installed. So you can activate it using:

***{{*** object.content**|markdown**:"codehilite" ***}}***

This will mark up the code as it’s output - but it won’t look any different! To make that happen, you’ll need to include a CSS file describing the style you want to apply.

For reasons best left to the imagination, pygments supplies the hilighting styles as [Python formatted files](http://pygments.org/docs/styles/) rather than CSS. In order to use them on your site therefore you need to convert them which you can do with the [pygmentize](http://pygments.org/docs/cmdline/) command:

pygmentize -S default -f html > default.css

Alternatively, you could save yourself a lot of bother and use the set available here:

https://github.com/richleland/pygments-css

Simply download the style you want (we’re using [colorful](https://github.com/richleland/pygments-css/blob/master/colorful.css)) and include it on your page. You should now have beautifully syntax highlighted code.

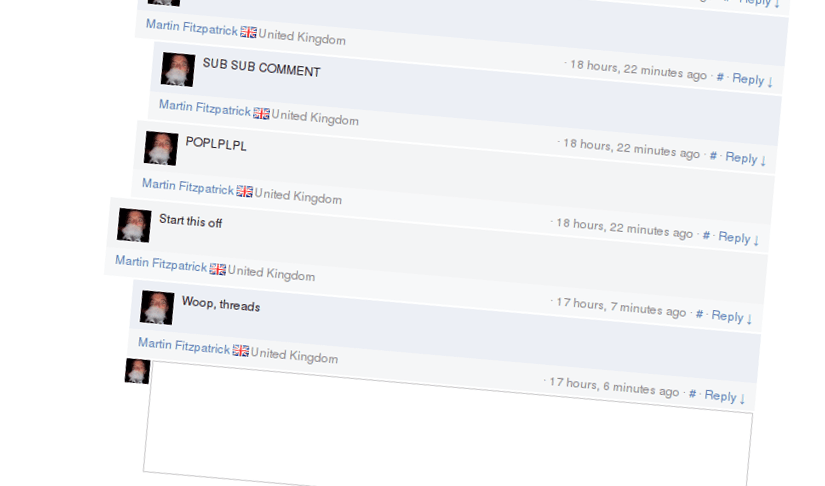
This method is based, with permission, on an original protocol available [here](http://freewisdom.org/projects/python-markdown/CodeHilite).

# Django Threaded Comments

by Martin Fitzpatrick — get latest articles [straight to your Inbox](http://mfitzp.us12.list-manage.com/subscribe/post?u=1e360ecc7f9998fd0be3ddb60&id=e16530b855).

Django ships with it’s own comments contrib app that provides commenting on arbitrary models. However this is a flat-comment system which doesn’t allow replying to comments. An app called django-threadedcomments exists but has not been updated for a number of versions and is broken. Other alternatives are less flexible than the Django commenting system itself.

Here we demonstrate a simple method to get threaded comments in Django using a combination of the inbuilt comment app and the mptt app.



Other solutions available at [django-threadedcomments](https://github.com/HonzaKral/django-threadedcomments) or [codeblogging](http://codeblogging.net/blogs/1/3/)

For this method you will also need [django-mptt](https://github.com/django-mptt/django-mptt/)

## Requirements

Django (with standard comments)

django-mptt

## Method

Django-mptt implements Modified Preorder Tree Traversal with your Django Models to produce working trees of Model instances. It achieves this by adding a number of hidden fields to your model. We’re we’re going to amend the inbuilt comments model (through class inheritence) to add these fields for django-mptt, then add some tweaks and hooks to make it all work together nicely.

The full [django-mptt documentation](http://django-mptt.github.com/django-mptt/) is worth a look.

To begin with you will need to install all the constituent parts. First install [django-mptt](https://github.com/django-mptt/django-mptt/) and add mptt and Django contrib comments apps to your INSTALLED\_APPS in your settings.py

Also you need to add your own comment app (that we’re creating here) to your INSTALLED\_APPS so you may as well do that now - we’ve called ours rather simply comments’.

INSTALLED\_APPS **=** (

**...**

'django.contrib.comments',

'mptt',

'comments',

**...**

)

Next add the following somewhere in your settings.py file:

COMMENTS\_APP **=** 'comments'

This tells Django where to go looking for comment-related tweaks you’ve created.

Now let’s create the app! In your ~/my\_project/apps/ folder create a new app with:

python manage**.**py startapp comments

This will create a folder with \_\_init\_\_.py, models.py, tests.py and views.py

Now we need to create the amended model, adding in the mptt fields. Thankfully django-mptt makes this remarkably easy, we only need to inherit from the MPTT model class and add a single field. So, here is our completed model (in models.py):

**from** django.contrib.comments.models **import** Comment

**from** mptt.models **import** MPTTModel, TreeForeignKey

**class** **MPTTComment**(MPTTModel, Comment):

""" Threaded comments - Add support for the parent comment store and MPTT traversal"""

*# a link to comment that is being replied, if one exists*

parent **=** TreeForeignKey('self', null**=**True, blank**=**True, related\_name**=**'children')

**class** **MPTTMeta**:

*# comments on one level will be ordered by date of creation*

order\_insertion\_by**=**['submit\_date']

**class** **Meta**:

ordering**=**['tree\_id','lft']

That’s it. Note we’ve added the submit\_date insertion for ordering.

Next we need to provide an amended form that will contain this parent field, so it can be correctly set and saved to the database. So create a new file called forms.py and enter the following:

**from** django **import** forms

**from** django.contrib.admin **import** widgets

**from** django.contrib.comments.forms **import** CommentForm

**from** models **import** MPTTComment

**class** **MPTTCommentForm**(CommentForm):

parent **=** forms**.**ModelChoiceField(queryset**=**MPTTComment**.**objects**.**all(), required**=**False, widget**=**forms**.**HiddenInput)

**def** **get\_comment\_model**(self):

*# Use our custom comment model instead of the built-in one.*

**return** MPTTComment

**def** **get\_comment\_create\_data**(self):

*# Use the data of the superclass, and add in the parent field field*

data **=** super(MPTTCommentForm, self)**.**get\_comment\_create\_data()

data['parent'] **=** self**.**cleaned\_data['parent']

**return** data

Django comments extensions work on a simple principle: you tell the comments app about what you’ve changed through the use of some simple hooks. Django then uses these hooks to lookup your altered models and forms and overrides the default set. So now that we’ve created our modified forms and models, we need to point Django at them.

To create these hooks you’ll need to edit the \_\_init\_\_.py file, so open it now. Enter the following:

**from** models **import** MPTTComment

**from** forms **import** MPTTCommentForm

**def** **get\_model**():

**return** MPTTComment

**def** **get\_form**():

**return** MPTTCommentForm

For more information on what is going on here have a look at [the Django documentation for customizing comments](https://docs.djangoproject.com/en/dev/ref/contrib/comments/custom/).

Now create a template for handling the comments. It is useful to have this as a separate include file, e.g. called \_comments.html that you can then call into another template. The example here expects the comment to be attached to an object called object so you may need to wrap the include {% with <yourmodel> as object %}{% include "\_comments.html" %}{% endwith %}.

***{%*** **load** comments ***%}***

***{%*** **load** mptt\_tags ***%}***

***{%*** **get\_comment\_list** for object **as** comments ***%}***

***{%*** **if** comments ***%}***

***{%*** **recursetree** comments ***%}***

<a name="c***{{*** node.id ***}}***"></a>

***{{*** node.comment ***}}***

***{{*** node.user ***}}***

***{{*** node.submit\_date**|timesince** ***}}*** ago

<a href="***{{*** object.get\_absolute\_url ***}}***#c***{{*** node.id ***}}***">#</a>

***{%*** **render\_comment\_form** for object ***%}***

*{# recursion! children of a given comment #}*

***{%*** **if** **not** node.is\_leaf\_node ***%}***

***{{*** children ***}}***

***{%*** **endif** ***%}***

***{%*** **endrecursetree** ***%}***

***{%*** **endif** ***%}***

***{%*** **render\_comment\_form** for object ***%}***

Finally, you’ll need to amend your comment form to include the parent\_id when posting a reply to another comment. So open the template in comments/form.html and enter:

***{%*** **load** comments i18n ***%}***

<form action="***{%*** **comment\_form\_target** ***%}***" method="post">***{%*** **csrf\_token** ***%}***

***{{*** form.object\_pk ***}}***

***{{*** form.content\_type ***}}***

***{{*** form.timestamp ***}}***

***{{*** form.security\_hash ***}}***

***{%*** **if** node.id ***%}***

<input type="hidden" name="parent" id="parent\_id" value="***{{*** node.id ***}}***" />

***{%*** **endif** ***%}***

***{{*** form.comment ***}}***

<input type="submit">

</form>

The key point is the inclusion of the parent field.

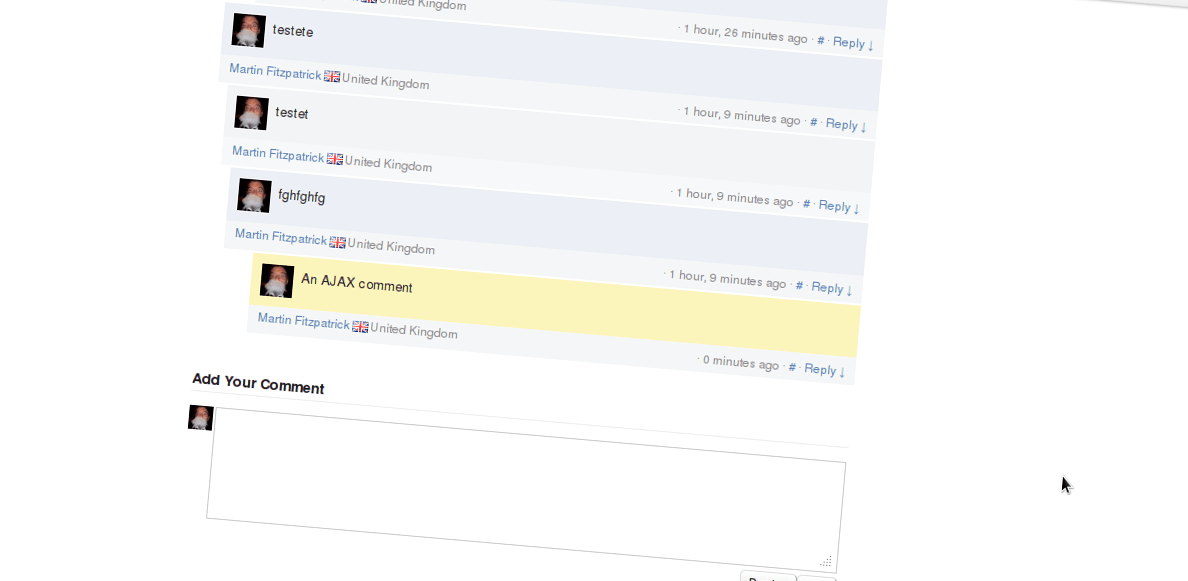
If you haven’t already you can copy the default Django template from the contrib folder into your own templates folder to allow you to edit it.

Fire it up! Have a test replying to comments both at the page, and threaded level. It should all work perfectly, but if not feel free to ask for help in our - threaded! - comments.

# Django AJAX threaded-comments using only jQuery

by Martin Fitzpatrick — get latest articles [straight to your Inbox](http://mfitzp.us12.list-manage.com/subscribe/post?u=1e360ecc7f9998fd0be3ddb60&id=e16530b855).

Upgrade threaded comments using mptt to allow AJAX submission of comments and update the template - without any backend hacking of Django. Everything is accomplished via Javascript and template tags.



## Method

Set up Django threaded comments using the [mptt method outlined previously](http://mfitzp.io/article/django-threaded-comments/).

First open up your comment form template form.html created in the threaded-comments tutorial and make the following changes:

<div class**=**"comment-form" id**=**"comment-form-{{ node.id }}" {% if node.id %}style**=**"display:none;"{% endif %}>

<form action**=**"{% comment\_form\_target %}" method**=**"post" id**=**"comment-form-{{ node.id }}f">

This simply ensures that the form and the wrapper both have unique ids on the page. This is required for the jQuery interactions to function reliably.

Next in the comment output template just beneath the {% recursetree comments %} wrap the comment output with the following div:

<div {% if request.REQUEST.c|add:"0" == node.id %}id**=**"newly\_posted\_comment"{% endif %}>

...comment content...

</div>

This looks for the c value set on the query string - as is done by the comment framework on the redirected-to page after submitting the comment. The odd looking c|add:"0" here is simply to coerce the string from the query string into an int for comparison with the node.id. Ugly but works.

The effect of this is giving the comment you posted the id newly\_posted\_comment - making it simple to find in the AJAX returned html.

You can also use this id to style the comment differently - e.g. highlighting it in some way. This will also work on non-AJAX requests.

Create a new template file in your comments templates folders named \_ajax\_comments.html. Add the following code below to this file and save. Note that you can alternatively put this in a .js file and include it as normal however you will need to hard-code the comment posting url.

{% load comments %}

<script type="text/javascript" charset="utf-8">

(function( $ ){

$.fn.bindPostCommentHandler = function() {

// We get passed a list of forms; iterate and get a unique id for each

// attach a submit trigger to handle saving and returning

this.each(function() {

//$(this).find('input.submit-preview').remove();

$(this).submit(function() {

commentform = this;

commentwrap = $(this).parent();

$.ajax({

type: "POST",

data: $(commentform).serialize(),

url: "{% comment\_form\_target %}",

cache: false,

dataType: "html",

success: function(html, textStatus) {

// Extract the form from the returned html

postedcomment = $(html).find('#newly\_posted\_comment');

$(commentform).replaceWith(postedcomment.html());

$(commentwrap).hide();

$(commentwrap).slideDown(600);

$(commentwrap).find('.comment-form form').bindPostCommentHandler();

},

error: function (XMLHttpRequest, textStatus, errorThrown) {

$(commentform).replaceWith('Your comment was unable to be posted at this time. We apologise for the inconvenience.');

}

});

return false;

});

}); //each

};

})( jQuery );

$(function() {

$('.comment-form form').bindPostCommentHandler();

});

</script>

If you’re interestd in what is happening in the code read on; if not you can safely skip this bit:

$('.comment-form form').bindPostCommentHandler();

$(this).submit(function() {

Together these bind our function onto the submit handler for all the comment forms on the page. This means when you hit submit it gets passed to our function below.

commentform = this;

commentwrap = $(this).parent();

In the function this is a reference to the form - we store this in variable commentform so it’s accessible within the ajax sucess function. We also store commentwrap which refers to the div wrapper of the comment form - it’s useful shorthand later.

$.ajax({

type: "POST",

data: $(commentform).serialize(),

url: "{% comment\_form\_target %}",

cache: false,

dataType: "html",

This is the submission handler for the AJAX request. We serialize the form commentform and submit it to {% comment\_form\_target %} and note that the data type returned will be HTML. This is the standard form submission URL we’re talking to so there will not be JSON/other things available.

success: function(html, textStatus) {

// Extract the form from the returned html

postedcomment = $(html).find('#newly\_posted\_comment');

$(commentform).replaceWith(postedcomment.html());

The success function is passed the html from the submission which - hopefully - will include the newly posted comment. So we use jQuery to find that element (which we’ve flagged with id newly\_posted\_comment) and replace the originating form with that content.

$(commentwrap).find('.comment-form form').bindPostCommentHandler();

Finally we bind our handler to this new comment form so the AJAX submission will keep on working.

error: function (XMLHttpRequest, textStatus, errorThrown) {

$(commentform).replaceWith('Your comment was unable to be posted at this time. We apologise for the inconvenience.');

This final bit is the error handler for when the AJAX submission goes wrong in any way. Here we replace the commentform with the error but you might instead want to insert the error before so the user can re-attempt submission.

Finally include the \_ajax\_comments.html on the template hosting your comments (or include the Javascript file if you chose that route). It needs to go into the head section of the html page so here I use {% block extrahead %} but your template may be different.

***{%*** **block** extrahead ***%}***

***{{*** block.super ***}}***

***{%*** **include** "comments/\_ajax\_comments.html" ***%}***

***{%*** **endblock** ***%}***

Refresh your page and attempt to submit a comment. You should find it magically sliding down from the comment above! No framework hacking neccessary.

The above is based on original code from [this article](http://ca.rroll.net/2009/05/10/improving-django-comments-user-experience-with-ajax/).

### Related posts

* [Django AJAX threaded-comments using only jQuery](https://martinfitzpatrick.name/article/django-ajax-threaded-comments-using-only-jquery/)
* [Django Threaded Comments](https://martinfitzpatrick.name/article/django-threaded-comments/)
* [Quickly serve a folder via Web Server](https://martinfitzpatrick.name/article/quickly-serve-a-folder-via-web-server/)
* [Highlight author's comments on WordPress blog](https://martinfitzpatrick.name/article/highlight-authors-comments-on-wordpress-blog/)
* [Display external RSS feeds on your WordPress blog](https://martinfitzpatrick.name/article/display-external-rss-feeds-on-your-wordpress-blog/)
* [Syntax highlighting with Django and Markdown](https://martinfitzpatrick.name/article/syntax-highlighting-with-django-and-markdown/)
* [Apache vs. the Slashdot Effect](https://martinfitzpatrick.name/article/apache-vs-the-slashdot-effect/)
* [Wooey](https://martinfitzpatrick.name/article/wooey/)