django-contact-form Documentation

Release 1.2

James Bennett

January 18, 2016

Contents

`django-contact-form` is a simple application providing simple, customizable contact-form functionality for Django-powered Web sites.

Basic functionality (collecting a name, email address and message) can be achieved uot of the box by setting up a few templates and adding one line to your site's root URLconf:

```
url(r'^contact/', include('contact_form.urls')),
```

For notes on getting started quickly, and on how to customize `django-contact-form`'s behavior, read through the full documentation below.

Contents:

CHAPTER 1

Installation guide

Before installing `django-contact-form`, you'll need to have a copy of Django already installed. For information on obtaining and installing Django, consult the Django download page, which offers convenient packaged downloads and installation instructions.

The 1.2 release of `django-registration` supports Django 1.8 and 1.9, on any Python version supported by those versions of Django:

- Django 1.8 suports Python 2.7, 3.2, 3.3, 3.4 and 3.5.
- Django 1.9 supports Python 2.7, 3.4 and 3.5.

Important: Python 3.2

Although Django 1.8 supports Python 3.2, and `django-registration` 1.2 supports it, many Python libraries supporting Python 3 impose a minimum requirement of Python 3.3 (due to conveniences added in Python 3.3 which make supporting Python 2 and 3 in the same codebase much simpler).

As a result, use of Python 3.2 is discouraged; Django 1.9 has already dropped support for it, and a future release of `django-registration` will drop Python 3.2 support as well.

1.1 Normal installation

The preferred method of installing `django-contact-form` is via `pip`, the standard Python package-installation tool. If you don't have `pip`, instructions are available for how to obtain and install it.

Once you have `pip`, simply type:

```
pip install django-contact-form
```

1.2 Manual installation

It's also possible to install `django-contact-form` manually. To do so, obtain the latest packaged version from the listing on the Python Package Index. Unpack the `.tar.gz` file, and run:

```
python setup.py install
```

Once you've installed `django-contact-form`, you can verify successful installation by opening a Python interpreter and typing `import contact_form`.

If the installation was successful, you'll simply get a fresh Python prompt. If you instead see an `ImportError`, check the configuration of your install tools and your Python import path to ensure `django-contact-form` installed into a location Python can import from.

1.3 Installing from a source checkout

The development repository for `django-contact-form` is at <https://github.com/ubernostrum/django-contact-form>. Presuming you have git installed, you can obtain a copy of the repository by typing:

```
git clone https://github.com/ubernostrum/django-contact-form.git
```

From there, you can use normal git commands to check out the specific revision you want, and install it using `python setup.py install`.

1.4 Basic configuration and use

Once you have Django and `django-contact-form` installed, check out *the quick start guide* to see how to get your contact form up and running.

CHAPTER 2

Quick start guide

First you'll need to have Django and `django-contact-form` installed; for details on that, see *the installation guide*.

Once that's done, you can start setting up `django-contact-form`. Since it doesn't provide any database models or use any other application-config mechanisms, you do *not* need to add `django-contact-form` to your `INSTALLED_APPS` setting; you can simply begin using it right away.

CHAPTER 3

URL configuration

The easiest way to set up the views in `django-contact-form` is to just use the provided URLconf, found at `contact_form.urls`. You can include it wherever you like in your site's URL configuration; for example, to have it live at the URL `/contact/`:

```
from django.conf.urls import include, url


urlpatterns = [
    # ... other URL patterns for your site ...
    url(r'^contact/', include('contact_form.urls')),
]
```

If you'll be using a custom form class, you'll need to manually set up your URLs so you can tell `django-contact-form` about your form class. For example:

```
from django.conf.urls import include, url
from django.views.generic import TemplateView

from contact_form.views import ContactFormView

from yourapp.forms import YourCustomFormClass


urlpatterns = [
    # ... other URL patterns for your site ...
    url(r'^contact/$',
        ContactFormView.as_view(
            form_class=YourCustomFormClass),
        name='contact_form'),
    url(r'^contact/sent/$',
        TemplateView.as_view(
            template_name='contact_form/contact_form_sent.html'),
        name='contact_form_sent'),
]
```

CHAPTER 4

Required templates

The two views above will need two templates to be created:

`contact_form/contact_form.html` This is used to display the contact form. It has a `RequestContext` (so any context processors will be applied), and also provides the form instance as the context variable `form`.

`contact_form/contact_form_sent.html` This is used after a successful form submission, to let the user know their message has been sent. It has a `RequestContext`, but provides no additional context variables of its own.

You'll also need to create at least two more templates to handle the rendering of the message: `contact_form/contact_form_subject.txt` for the subject line of the email to send, and `contact_form/contact_form.txt` for the body (note that the file extension for these is `.txt`, not `.html`!). Both of these will receive a `RequestContext` with a set of variables named for the fields of the form (by default: `name`, `email` and `body`), as well as one more variable: `site`, representing the current site (either a `Site` or `RequestSite` instance, depending on whether Django's sites framework is installed).

Warning: Subject must be a single line
In order to prevent header injection attacks, the subject *must* be only a single line of text, and Django's email framework will reject any attempt to send an email with a multi-line subject. So it's a good idea to ensure your `contact_form_subject.txt` template only produces a single line of output when rendered; as a precaution, however, `django-contact-form` will split the output of this template at line breaks, then forcibly re-join it into a single line of text.

CHAPTER 5

The ContactForm class

class contact_form.forms.**ContactForm**

The base contact form class from which all contact form classes should inherit.

If you don't need any custom functionality, you can simply use this form to provide basic contact functionality; it will collect name, email address and message.

The *ContactFormView* included in this application knows how to work with this form and can handle many types of subclasses as well (see below for a discussion of the important points), so in many cases it will be all that you need. If you'd like to use this form or a subclass of it from one of your own views, just do the following:

1. When you instantiate the form, pass the current `HttpRequest` object as the keyword argument `request`; this is used internally by the base implementation, and also made available so that subclasses can add functionality which relies on inspecting the request (such as spam filtering).
2. To send the message, call the form's `save` method, which accepts the keyword argument `fail_silently` and defaults it to `False`. This argument is passed directly to Django's `send_mail()` function, and allows you to suppress or raise exceptions as needed for debugging. The `save` method has no return value.

Other than that, treat it like any other form; validity checks and validated data are handled normally, through the `is_valid()` method and the `cleaned_data` dictionary.

Under the hood, this form uses a somewhat abstracted interface in order to make it easier to subclass and add functionality.

These attributes play a role in determining behavior:

from_email
: The email address to use in the `From:` header of the message. This can also be implemented as a method named `from_email()`, in which case it will be called when constructing the message. By default, this is the value of the setting `DEFAULT_FROM_EMAIL`.

recipient_list
: The list of recipients for the message. This can also be implemented as a method named `recipient_list()`, in which case it will be called when constructing the message. By default, this is the email addresses specified in the setting `MANAGERS`.

subject_template_name
: The name of the template to use when rendering the subject line of the message. By default, this is `contact_form/contact_form_subject.txt`.

template_name
: The name of the template to use when rendering the body of the message. By default, this is `contact_form/contact_form.txt`.

And two methods are involved in actually producing the contents of the message to send:

message()
: Returns the body of the message to send. By default, this is accomplished by rendering the template name specified in *template_name*.

subject()
: Returns the subject line of the message to send. By default, this is accomplished by rendering the template name specified in *subject_template_name*.

Finally, the message itself is generated by the following two methods:

get_message_dict()
: This method loops through *from_email*, *recipient_list*, *message()* and *subject()*, collecting those parts into a dictionary with keys corresponding to the arguments to Django's `send_mail` function, then returns the dictionary. Overriding this allows essentially unlimited customization of how the message is generated.

get_context()
: For methods which render portions of the message using templates (by default, *message()* and *subject()*), generates the context used by those templates. The default context will be a `RequestContext` (using the current HTTP request, so user information is available), plus the contents of the form's `cleaned_data` dictionary, and one additional variable:

  site If `django.contrib.sites` is installed, the currently-active `Site` object. Otherwise, a `RequestSite` object generated from the request.

Meanwhile, the following attributes/methods generally should not be overridden; doing so may interfere with functionality, may not accomplish what you want, and generally any desired customization can be accomplished in a more straightforward way through overriding one of the attributes/methods listed above.

request
: The `HttpRequest` object representing the current request. This is set automatically in `__init__()`, and is used both to generate a `RequestContext` for the templates and to allow subclasses to engage in request-specific behavior.

save()
: If the form has data and is valid, will actually send the email, by calling *get_message_dict()* and passing the result to Django's `send_mail` function.

Note that subclasses which override `__init__` or *save()* need to accept `*args` and `**kwargs`, and pass them via `super`, in order to preserve behavior (each of those methods accepts at least one additional argument, and this application expects and requires them to do so).

CHAPTER 6

Built-in views

class contact_form.views.**ContactFormView**

The base view class from which most custom contact-form views should inherit. If you don't need any custom functionality, and are content with the default *ContactForm* class, you can also just use it as-is (and the provided URLConf, contact_form.urls, does exactly this).

This is a subclass of Django's FormView, so refer to the Django documentation for a list of attributes/methods which can be overridden to customize behavior.

One non-standard attribute is defined here:

recipient_list

The list of email addresses to send mail to. If not specified, defaults to the *recipient_list* of the form.

Additionally, the following standard (from FormView) methods and attributes are commonly useful to override (all attributes below can also be passed to as_view() in the URLconf, permitting customization without the need to write a full custom subclass of ContactFormView):

form_class

The form class to use. By default, will be *ContactForm*. This can also be overridden as a method named form_class(); this permits, for example, per-request customization (by inspecting attributes of self.request).

template_name

The template to use when rendering the form. By default, will be contact_form/contact_form.html.

get_success_url()

The URL to redirect to after successful form submission. By default, this is the named URL contact_form.sent. In the default URLconf provided with django-contact-form, that URL is mapped to TemplateView rendering the template contact_form/contact_form_sent.html.

get_form_kwargs()

Returns additional keyword arguments (as a dictionary) to pass to the form class on initialization.

By default, this will return a dictionary containing the current HttpRequest (as the key request) and, if *recipient_list* was defined, its value (as the key recipient_list).

> **Warning:** If you override get_form_kwargs(), you **must** ensure that, at the very least, the keyword argument request is still provided, or ContactForm initialization will raise TypeError. The simplest approach is to use super() to call the base implementation in ContactFormView, and modify the dictionary it returns.

Warning: Implementing `form_invalid()`
To work around a potential performance issue in Django 1.9, `ContactFormView` implements the `form_invalid()` method. If you choose to override `form_invalid()` in a subclass of `ContactFormView`, be sure to read the implementation and comments in the source code of `django-contact-form` first. Note that Django 1.9.1, once released, will not be affected by this bug.

CHAPTER 7

Frequently asked questions

The following notes answer some common questions, and may be useful to you when installing, configuring or using `django-contact-form`.

7.1 What versions of Django are supported?

As of `django-contact-form` 1.2, Django 1.8 and 1.9 are supported.

7.2 What versions of Python are supported?

As of 1.2, `django-contact-form` supports Python 2.7, 3.2, 3.3, 3.4, and 3.5. Note that Django 1.9 dropped support for Python 3.2 and 3.3, and a future release of `django-contact-form` will drop Python 3.2 support.

7.3 What license is `django-contact-form` under?

`django-contact-form` is offered under a three-clause BSD-style license; this is an OSI-approved open-source license, and allows you a large degree of freedom in modifiying and redistributing the code. For the full terms, see the file `LICENSE` which came with your copy of django-contact-form; if you did not receive a copy of this file, you can view it online at <https://github.com/ubernostrum/django-contact-form/blob/master/LICENSE>.

7.4 Why aren't there any default templates I can use?

Usable default templates, for an application designed to be widely reused, are essentially impossible to produce; variations in site design, block structure, etc. cannot be reliably accounted for. As such, `django-contact-form` provides bare-bones (i.e., containing no HTML structure whatsoever) templates in its source distribution to enable running tests, and otherwise simply provides good documentation of all required templates and the context made available to them.

7.5 What happened to the spam-filtering form in previous versions?

Older versions of `django-contact-form` shipped a subclass of *`ContactForm`* which used the Akismet web service to identify and reject spam submissions.

Unfortunately, the Akismet Python library – required in order to use such a class – does not currently support all versions of Python on which `django-contact-form` is supported, meaning it cannot be included in `django-contact-form` by default. The author of `django-contact-form` is working on producing a version of the Akismet library compatible with Python 3, but it was not yet ready as of the release of `django-contact-form` 1.2.

7.6 Why am I getting a bunch of `BadHeaderError` exceptions?

Most likely, you have an error in your *`ContactForm`* subclass. Specifically, one or more of *`from_email`*, *`recipient_list`* or *`subject()`* are returning values which contain newlines.

As a security precaution against email header injection attacks (which allow spammers and other malicious users to manipulate email and potentially cause automated systems to send mail to unintended recipients), Django's email-sending framework does not permit newlines in message headers. `BadHeaderError` is the exception Django raises when a newline is detected in a header.

Note that this only applies to the headers of an email message; the message body can (and usually does) contain newlines.

7.7 I found a bug or want to make an improvement!

The canonical development repository for `django-contact-form` is online at <https://github.com/ubernostrum/django-contact-form>. Issues and pull requests can both be filed there.

If you'd like to contribute to `django-contact-form`, that's great! Just please remember that pull requests should include tests and documentation for any changes made, and that following PEP 8 is mandatory. Pull requests without documentation won't be merged, and PEP 8 style violations or test coverage below 100% are both configured to break the build.

C

C

F

G

M

R

S

T