django-contact-form Documentation

Release 1.1

James Bennett

January 17, 2016

Contents

Providing some sort of contact or feedback form for soliciting information from site visitors is a common need in web development, and writing a contact form and associated handler view, while relatively straightforward to do with Django, can be a tedious and repetitive task.

This application aims to remove or reduce that tedium and repetition by providing simple, extensible contact-form functionality for Django-powered sites.

In the simplest case, all that's required is a bit of configuration and a few templates, and one pattern in your URLConf:

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

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 1.1 release of `django-contact-form` supports Django 1.7 and 1.8, on any of Python 2.7, 3.3 or 3.4. Older versions of Django and/or Python may work, but are not tested or officially supported. It is expected that `django-contact-form` 1.1 will be compatible with Python 3.5 once released (as of the release of `django-contact-form` 1.1, Python 3.5 was in beta testing).

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 installed, only a small amount of setup is required to use `django-contact-form`. First, you'll need to make sure you've specified the appropriate settings for Django to send email. Most commonly, this will be `EMAIL_HOST`, `EMAIL_PORT`, `EMAIL_HOST_USER` and `EMAIL_HOST_PASSWORD`.

You'll also want to make sure `django-contact-form` sends mail from the correct address, and sends to the correct address(es). Two standard Django settings control this:

- By default, the `From:` header of all emails sent by `django-contact-form` will be whatever email address is specified in `DEFAULT_FROM_EMAIL`.
- By default, the recipient list for emails sent by `django-contact-form` will be the email addresses specified in `MANAGERS`.

If you'd prefer something else, this behavior is configurable; see *the form documentation* for details on how to customize the email addresses used.

1.4.1 Templates

The following templates are required by the default setup of `django-contact-form`, so you'll need to create them:

- `contact_form/contact_form.html` is the template which actually renders the contact form. Important context variables are:

  `form` The contact form instance.

- `contact_form/contact_form_sent.html` is the template rendered after a message is successfully sent through the contact form. It has no specific context variables, beyond whatever's supplied by the context processors in use on your site.

Additionally, the generated email makes use of two templates: `contact_form/contact_form_subject.txt` will be rendered to obtain the subject line, and `contact_form/contact_form.txt` will be rendered to obtain the body of the email. These templates use `RequestContext`, so any context processors will be applied, and have the following additional context:

`site` The current site. Either a `Site` instance if `django.contrib.sites` is installed, or a `RequestSite` instance if not.

`body` The body of the message the user entered into the contact form.

`email` The email address the user supplied to the contact form.

`name` The name the user supplied to the contact form.

1.4.2 URL configuration

Once you've got settings and templates set up, all that's left is to configure your URLs to point to the `django-contact-form` views. A URLconf – `contact_form.urls` – is provided with

`django-contact-form`, which will wire up these views with default behavior; to make use of it, simply include it at whatever point in your URL hierarchy you'd like your contact form to live. For example, to place it at `/contact/`:

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

CHAPTER 2

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 to the constructor 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 `send_mail`, 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 3

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:

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.

CHAPTER 4

Frequently asked questions

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

4.1 What versions of Django are supported?

As of `django-contact-form` 1.1, Django 1.7 and 1.8 are supported.

4.2 What versions of Python are supported?

As of 1.1, `django-contact-form` supports Python 2.7, 3.3, and 3.4. It is anticipated that when Python 3.5 is released, `django-contact-form` 1.1 will be compatible with it.

4.3 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.

4.4 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` 1.1.

4.5 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.

C

C

F

G

M

R

S

T