Django settings

A Django settings file contains all the configuration of your Django installation. This document explains how settings work and which settings are available.

The basics

A settings file is just a Python module with module-level variables.

Here are a couple of example settings:

```
ALLOWED_HOSTS = ["www.example.com"]
DEBUG = False
DEFAULT_FROM_EMAIL = "webmaster@example.com"
```

Note: If you set DEBUG to `False`, you also need to properly set the ALLOWED_HOSTS setting.

Because a settings file is a Python module, the following apply:

- It doesn't allow for Python syntax errors.
- It can assign settings dynamically using normal Python syntax. For example:

  ```
  MY_SETTING = [str(i) for i in range(30)]
  ```

- It can import values from other settings files.

Designating the settings

```
DJANGO_SETTINGS_MODULE
```

When you use Django, you have to tell it which settings you're using. Do this by using an environment variable, DJANGO_SETTINGS_MODULE.

The value of DJANGO_SETTINGS_MODULE should be in Python path syntax, e.g. `mysite.settings`. Note that the settings module should be on the Python sys.path.

The `django-admin` utility

When using django-admin, you can either set the environment variable once, or explicitly pass in the settings module each time you run the utility.

Example (Unix Bash shell):

```
export DJANGO_SETTINGS_MODULE=mysite.settings
django-admin runserver
```

Example (Windows shell):

```
set DJANGO_SETTINGS_MODULE=mysite.settings
django-admin runserver
```

Use the `--settings` command-line argument to specify the settings manually:

```
django-admin runserver --settings=mysite.settings
```

On the server (`mod_wsgi`)

In your live server environment, you'll need to tell your WSGI application what settings file to use. Do that with `os.environ`:

```
import os

os.environ["DJANGO_SETTINGS_MODULE"] = "mysite.settings"
```

Read the Django mod_wsgi documentation for more information and other common elements to a Django WSGI application.

Default settings

A Django settings file doesn't have to define any settings if it doesn't need to. Each setting has a sensible default value. These defaults live in the module django/conf/global_settings.py.

Here's the algorithm Django uses in compiling settings:

- Load settings from `global_settings.py`.
- Load settings from the specified settings file, overriding the global settings as necessary.

Note that a settings file should *not* import from `global_settings`, because that's redundant.

Seeing which settings you've changed

The command `python manage.py diffsettings` displays differences between the current settings file and Django's default settings.

For more, see the `diffsettings` documentation.

Using settings in Python code

In your Django apps, use settings by importing the object `django.conf.settings`. Example:

```
from django.conf import settings

if settings.DEBUG:
    # Do something
    ...
```

Note that `django.conf.settings` isn't a module – it's an object. So importing individual settings is not possible:

```
from django.conf.settings import DEBUG  # This won't work.
```

Also note that your code should *not* import from either `global_settings` or your own settings file. `django.conf.settings` abstracts the concepts of default settings and site-specific settings; it presents a single interface. It also decouples the code that uses settings from the location of your settings.

Altering settings at runtime

You shouldn't alter settings in your applications at runtime. For example, don't do this in a view:

```
from django.conf import settings

settings.DEBUG = True  # Don't do this!
```

The only place you should assign to settings is in a settings file.

Security

Because a settings file contains sensitive information, such as the database password, you should make every attempt to limit access to it. For example, change its file permissions so that only you and your web server's user can read it. This is especially important in a shared-hosting environment.

Available settings

For a full list of available settings, see the settings reference.

Creating your own settings

There's nothing stopping you from creating your own settings, for your own Django apps, but follow these guidelines:

- Setting names must be all uppercase.
- Don't reinvent an already-existing setting.

For settings that are sequences, Django itself uses lists, but this is only a convention.

Using settings without setting DJANGO_SETTINGS_MODULE

In some cases, you might want to bypass the `DJANGO_SETTINGS_MODULE` environment variable. For example, if you're using the template system by itself, you likely don't want to have to set up an environment variable pointing to a settings module.

In these cases, you can configure Django's settings manually. Do this by calling:

```
django.conf.settings.configure(default_settings, **settings)
```

Example:

```
from django.conf import settings

settings.configure(DEBUG=True)
```

Pass `configure()` as many keyword arguments as you'd like, with each keyword argument representing a setting and its value. Each argument name should be all uppercase, with the same name as the settings described above. If a particular setting is not passed to `configure()` and is needed at some later point, Django will use the default setting value.

Configuring Django in this fashion is mostly necessary – and, indeed, recommended – when you're using a piece of the framework inside a larger application.

Consequently, when configured via `settings.configure()`, Django will not make any modifications to the process environment variables (see the documentation of `TIME_ZONE` for why this would normally occur). It's assumed that you're already in full control of your environment in these cases.

Custom default settings

If you'd like default values to come from somewhere other than `django.conf.global_settings`, you can pass in a module or class that provides the default settings as the `default_settings` argument (or as the first positional argument) in the call to `configure()`.

In this example, default settings are taken from `myapp_defaults`, and the `DEBUG` setting is set to `True`, regardless of its value in `myapp_defaults`:

```
from django.conf import settings
from myapp import myapp_defaults

settings.configure(default_settings=myapp_defaults, DEBUG=True)
```

The following example, which uses `myapp_defaults` as a positional argument, is equivalent:

```
settings.configure(myapp_defaults, DEBUG=True)
```

Normally, you will not need to override the defaults in this fashion. The Django defaults are sufficiently tame that you can safely use them. Be aware that if you do pass in a new default module, it entirely *replaces* the Django defaults, so you must specify a value for every possible setting that might be used in the code you are importing. Check in `django.conf.settings.global_settings` for the full list.

Either `configure()` or is required `DJANGO_SETTINGS_MODULE`

If you're not setting the `DJANGO_SETTINGS_MODULE` environment variable, you *must* call `configure()` at some point before using any code that reads settings.

If you don't set `DJANGO_SETTINGS_MODULE` and don't call `configure()`, Django will raise an `ImportError` exception the first time a setting is accessed.

If you set `DJANGO_SETTINGS_MODULE`, access settings values somehow, *then* call `configure()`, Django will raise a `RuntimeError` indicating that settings have already been configured. There is a property for this purpose:

```
django.conf.settings.configured
```

For example:

```
from django.conf import settings

if not settings.configured:
    settings.configure(myapp_defaults, DEBUG=True)
```

Also, it's an error to call `configure()` more than once, or to call `configure()` after any setting has been accessed.

It boils down to this: Use exactly one of either `configure()` or `DJANGO_SETTINGS_MODULE`. Not both, and not neither.

Calling `django.setup()` is required for "standalone" Django usage

If you're using components of Django "standalone" – for example, writing a Python script which loads some Django templates and renders them, or uses the ORM to fetch some data – there's one more step you'll need in addition to configuring settings.

After you've either set `DJANGO_SETTINGS_MODULE` or called `configure()`, you'll need to call `django.setup()` to load your settings and populate Django's application registry. For example:

```
import django
from django.conf import settings
from myapp import myapp_defaults

settings.configure(default_settings=myapp_defaults, DEBUG=True)
django.setup()

# Now this script or any imported module can use any part of Django it needs.
from myapp import models
```

Note that calling `django.setup()` is only necessary if your code is truly standalone. When invoked by your web server, or through django-admin, Django will handle this for you.

`django.setup()` **may only be called once.**: Therefore, avoid putting reusable application logic in standalone scripts so that you have to import from the script elsewhere in your application. If you can't avoid that, put the call to `django.setup()` inside an `if` block:

```
if __name__ == "__main__":
    import django

    django.setup()
```

See also:

The Settings Reference

Contains the complete list of core and contrib app settings.