Djrill Documentation

Release 1.4.0

Djrill contributors (see AUTHORS.txt)

May 13, 2015

Contents

Version 1.4.0

Djrill integrates the Mandrill transactional email service into Django.

In general, Djrill "just works" with Django's built-in `django.core.mail` package. It includes:

- Support for HTML, attachments, extra headers, and other features of Django's built-in email

- Mandrill-specific extensions like tags, metadata, tracking, and MailChimp templates

- Optional support for Mandrill inbound email and other webhook notifications, via Django signals

- An optional Django admin interface

Djrill is released under the BSD license. It is tested against Django 1.3–1.8 (including Python 3 with Django 1.6+, and PyPy support with Django 1.5+). Djrill uses semantic versioning.

Documentation

1.1 Djrill 1-2-3

1. Install Djrill from PyPI:

```
$ pip install djrill
```

2. Edit your project's `settings.py`:

```python
INSTALLED_APPS = (
    ...
    "djrill"
)


MANDRILL_API_KEY = "<your Mandrill key>"
EMAIL_BACKEND = "djrill.mail.backends.djrill.DjrillBackend"
DEFAULT_FROM_EMAIL = "you@example.com"  # if you don't already have this in settings
```

3. Now the regular Django email functions will send through Mandrill:

```python
from django.core.mail import send_mail

send_mail("It works!", "This will get sent through Mandrill",
    "Djrill Sender <djrill@example.com>", ["to@example.com"])
```

You could send an HTML message, complete with custom Mandrill tags and metadata:

```python
from django.core.mail import EmailMultiAlternatives

msg = EmailMultiAlternatives(
    subject="Djrill Message",
    body="This is the text email body",
    from_email="Djrill Sender <djrill@example.com>",
    to=["Recipient One <someone@example.com>", "another.person@example.com"],
    headers={'Reply-To': "Service <support@example.com>"} # optional extra headers
)
msg.attach_alternative("<p>This is the HTML email body</p>", "text/html")

# Optional Mandrill-specific extensions:
msg.tags = ["one tag", "two tag", "red tag", "blue tag"]
msg.metadata = {'user_id': "8675309"}

# Send it:
msg.send()
```

1.2 Installation

It's easiest to install Djrill from PyPI:

```
$ pip install djrill
```

If you decide to install Djrill some other way, you'll also need to install its one dependency (other than Django, of course): the requests library from Kenneth Reitz.

1.2.1 Configuration

In your project's `settings.py`:

1. Add `djrill` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    "djrill"
)
```

2. Add the following line, substituting your own `MANDRILL_API_KEY`:

```
MANDRILL_API_KEY = "brack3t-is-awesome"
```

3. Override your existing `EMAIL_BACKEND` with the following line:

```
EMAIL_BACKEND = "djrill.mail.backends.djrill.DjrillBackend"
```

Also, if you don't already have a `DEFAULT_FROM_EMAIL` in settings, this is a good time to add one. (Django's default is "webmaster@localhost", which won't work with Mandrill.)

1.2.2 Mandrill Webhooks (Optional)

Djrill includes optional support for Mandrill webhooks, including inbound email. See the Djrill *webhooks* section for configuration details.

1.2.3 Mandrill Subaccounts (Optional)

If you are using Mandrill's subaccounts feature, you can globally set the subaccount for all messages sent through Djrill:

```
MANDRILL_SUBACCOUNT = "client-347"
```

(You can also set or override the *subaccount* on each individual message, with *Mandrill-specific sending options*.)

New in version 1.0: MANDRILL_SUBACCOUNT global setting

1.2.4 Admin (Optional)

Djrill includes an optional Django admin interface, which allows you to:

- Check the status of your Mandrill API connection
- See stats on email senders, tags and urls

If you want to enable the Djrill admin interface, edit your base `urls.py`:

```
...
from django.contrib import admin

from djrill import DjrillAdminSite

admin.site = DjrillAdminSite()
admin.autodiscover()
...

urlpatterns = [
    ...
    url(r'^admin/', include(admin.site.urls)),
]
```

If you are on **Django 1.7 or later,** you will also need to change the config used by the django.contrib.admin app in your settings.py:

```
...
INSTALLED_APPS = (
    # For Django 1.7+, use SimpleAdminConfig because we'll call autodiscover...
    'django.contrib.admin.apps.SimpleAdminConfig',  # instead of 'django.contrib.admin'
    ...
    'djrill',
    ...
)
...
```

1.3 Sending Mail

Djrill handles **all** outgoing email sent through Django's standard django.core.mail package, including send_mail(), send_mass_mail(), the EmailMessage class, and even mail_admins().

If you'd like to selectively send only some messages through Mandrill, there is a way to *use multiple email backends*.

1.3.1 Django Email Support

Djrill supports most of the functionality of Django's EmailMessage and EmailMultiAlternatives classes.

Some notes and limitations:

Display Names All email addresses (from, to, cc, bcc) can be simple ("email@example.com") or can include a display name ("Real Name <email@example.com>").

CC and BCC Recipients Djrill properly identifies "cc" and "bcc" recipients to Mandrill.

Note that you may need to set the Mandrill option *preserve_recipients* to True if you want recipients to be able to see who else was included in the "to" list.

Changed in version 0.9: Previously, Djrill (and Mandrill) didn't distinguish "cc" from "to", and allowed only a single "bcc" recipient.

HTML/Alternative Parts To include an HTML version of a message, use attach_alternative():

```
from django.core.mail import EmailMultiAlternatives

msg = EmailMultiAlternatives("Subject", "text body",
                             "from@example.com", ["to@example.com"])
msg.attach_alternative("<html>html body</html>", "text/html")
```

Djrill allows a maximum of one `attach_alternative()` on a message, and it must be `mimetype="text/html"`. Otherwise, Djrill will raise *`NotSupportedByMandrillError`* when you attempt to send the message. (Mandrill doesn't support sending multiple html alternative parts, or any non-html alternatives.)

Attachments Djrill will send a message's attachments. (Note that Mandrill may impose limits on size and type of attachments.)

Also, if an image attachment has a Content-ID header, Djrill will tell Mandrill to treat that as an embedded image rather than an ordinary attachment. (For an example, see `test_embedded_images()` in `tests/test_mandrill_send.py`.)

New in version 0.3: Attachments

Changed in version 0.4: Special handling for embedded images

Headers Djrill accepts additional headers and passes them along to Mandrill:

```
msg = EmailMessage( ...
    headers={'Reply-To': "reply@example.com", 'List-Unsubscribe': "..."}
)
```

Changed in version 0.9: In earlier versions, Djrill only allowed `Reply-To` and `X-*` headers, matching previous Mandrill API restrictions.

Changed in version 1.4: Djrill also supports the `reply_to` param added to `EmailMessage` in Django 1.8. (If you provide *both* a 'Reply-To' header and the `reply_to` param, the header will take precedence.)

1.3.2 Mandrill-Specific Options

Most of the options from the Mandrill [messages/send API](#) `message` struct can be set directly on an `EmailMessage` (or subclass) object:

important
> `Boolean`: whether Mandrill should send this message ahead of non-important ones.
>
> New in version 0.7.

track_opens
> `Boolean`: whether Mandrill should enable open-tracking for this message. Default from your Mandrill account settings.

```
message.track_opens = True
```

track_clicks
> `Boolean`: whether Mandrill should enable click-tracking for this message. Default from your Mandrill account settings.
>
> ---
> **Note:** Mandrill has an option to track clicks in HTML email but not plaintext, but it's *only* available in your Mandrill account settings. If you want to use that option, set it at Mandrill, and *don't* set the `track_clicks` attribute here.
>
> ---

auto_text
> `Boolean`: whether Mandrill should automatically generate a text body from the HTML. Default from your Mandrill account settings.

auto_html
> `Boolean`: whether Mandrill should automatically generate an HTML body from the plaintext. Default from your Mandrill account settings.

inline_css
    `Boolean`: whether Mandrill should inline CSS styles in the HTML. Default from your Mandrill account settings.

    New in version 0.4.

url_strip_qs
    `Boolean`: whether Mandrill should ignore any query parameters when aggregating URL tracking data. Default from your Mandrill account settings.

preserve_recipients
    `Boolean`: whether Mandrill should include all recipients in the "to" message header. Default from your Mandrill account settings.

view_content_link
    `Boolean`: set False on sensitive messages to instruct Mandrill not to log the content.

    New in version 0.7.

tracking_domain
    `str`: domain Mandrill should use to rewrite tracked links and host tracking pixels for this message. Useful if you send email from multiple domains. Default from your Mandrill account settings.

signing_domain
    `str`: domain Mandrill should use for DKIM signing and SPF on this message. Useful if you send email from multiple domains. Default from your Mandrill account settings.

return_path_domain
    `str`: domain Mandrill should use for the message's return-path.

    New in version 0.7.

merge_language
    `str`: the merge tag language if using merge tags – e.g., "mailchimp" or "handlebars". Default from your Mandrill account settings.

    New in version 1.3.

global_merge_vars
    `dict`: merge variables to use for all recipients (most useful with *Mandrill Templates*).

```
message.global_merge_vars = {'company': "ACME", 'offer': "10% off"}
```

    Merge data must be strings or other JSON-serializable types. (See *Formatting Merge Data* for details.)

merge_vars
    `dict`: per-recipient merge variables (most useful with *Mandrill Templates*). The keys in the dict are the recipient email addresses, and the values are dicts of merge vars for each recipient:

```
message.merge_vars = {
    'wiley@example.com': {'offer': "15% off anvils"},
    'rr@example.com':    {'offer': "instant tunnel paint"}
}
```

    Merge data must be strings or other JSON-serializable types. (See *Formatting Merge Data* for details.)

tags
    `list` of `str`: tags to apply to the message, for filtering reports in the Mandrill dashboard. (Note that Mandrill prohibits tags longer than 50 characters or starting with underscores.)

```
message.tags = ["Order Confirmation", "Test Variant A"]
```

subaccount

> str: the ID of one of your subaccounts to use for sending this message. (The subaccount on an individual message will override any global *MANDRILL_SUBACCOUNT* setting.)

> New in version 0.7.

google_analytics_domains

> list of str: domain names for links where Mandrill should add Google Analytics tracking parameters.

```
message.google_analytics_domains = ["example.com"]
```

google_analytics_campaign

> str or list of str: the utm_campaign tracking parameter to attach to links when adding Google Analytics tracking. (Mandrill defaults to the message's from_email as the campaign name.)

metadata

> dict: metadata values Mandrill should store with the message for later search and retrieval.

```
message.metadata = {'customer': customer.id, 'order': order.reference_number}
```

> Mandrill restricts metadata keys to alphanumeric characters and underscore, and metadata values to numbers, strings, boolean values, and None (null).

recipient_metadata

> dict: per-recipient metadata values. Keys are the recipient email addresses, and values are dicts of metadata for each recipient (similar to *merge_vars*)

> Mandrill restricts metadata keys to alphanumeric characters and underscore, and metadata values to numbers, strings, boolean values, and None (null).

async

> Boolean: whether Mandrill should use an async mode optimized for bulk sending.

> New in version 0.7.

ip_pool

> str: name of one of your Mandrill dedicated IP pools to use for sending this message.

> New in version 0.7.

send_at

> datetime or date or str: instructs Mandrill to delay sending this message until the specified time. (Djrill allows timezone-aware Python datetimes, and converts them to UTC for Mandrill. Timezone-naive datetimes are assumed to be UTC.)

> New in version 0.7.

These Mandrill-specific properties work with *any* EmailMessage-derived object, so you can use them with many other apps that add Django mail functionality.

If you have questions about the python syntax for any of these properties, see DjrillMandrillFeatureTests in tests/test_mandrill_send.py for examples.

1.3.3 Mandrill Response

A mandrill_response property is added to each EmailMessage that you send. This allows you to retrieve message ids, initial status information and more.

For an EmailMessage that is successfully sent to one or more email addresses, mandrill_response will be set to a list of dict, where each entry has info for one email address. See the Mandrill docs for the messages/send API for full details.

For example, to get the Mandrill message id for a sent email you might do this:

```
msg = EmailMultiAlternatives(subject="subject", body="body",
                             from_email="sender@example.com",to=["someone@example.com"])
msg.send()
response = msg.mandrill_response[0]
mandrill_id = response['_id']
```

For this example, msg.mandrill_response might look like this:

```
msg.mandrill_response = [
    {
        "email": "someone@example.com",
        "status": "sent",
        "_id": "abc123abc123abc123abc123abc123"
    }
]
```

If an error is returned by Mandrill while sending the message then `mandrill_response` will be set to None.

New in version 0.8: mandrill_response available for sent messages

1.3.4 Exceptions

New in version 0.3: Djrill-specific exceptions

exception djrill.**NotSupportedByMandrillError**

> If the email tries to use features that aren't supported by Mandrill, the send call will raise a `NotSupportedByMandrillError` exception (a subclass of `ValueError`).

exception djrill.**MandrillAPIError**

> If the Mandrill API fails or returns an error response, the send call will raise a `MandrillAPIError` exception (a subclass of `requests.HTTPError`). The exception's `status_code` and `response` attributes may help explain what went wrong. (Tip: you can also check Mandrill's API error log to view the full API request and error response.)

1.4 Sending Template Mail

1.4.1 Mandrill Templates

New in version 0.3: Mandrill template support

To use a *Mandrill* (MailChimp) template stored in your Mandrill account, set a `template_name` and (optionally) `template_content` on your `EmailMessage` object:

```python
from django.core.mail import EmailMessage

msg = EmailMessage(subject="Shipped!", from_email="store@example.com",
                   to=["customer@example.com", "accounting@example.com"])
msg.template_name = "SHIPPING_NOTICE"            # A Mandrill template name
msg.template_content = {                         # Content blocks to fill in
    'TRACKING_BLOCK': "<a href='.../*|TRACKINGNO|*'>track it</a>"
}
msg.global_merge_vars = {                        # Merge tags in your template
    'ORDERNO': "12345", 'TRACKINGNO': "1Z987"
}
```

```
msg.merge_vars = {                              # Per-recipient merge tags
    'accounting@example.com': {'NAME': "Pat"},
    'customer@example.com':   {'NAME': "Kim"}
}
msg.send()
```

If `template_name` is set, Djrill will use Mandrill's messages/send-template API, and will ignore any `body` text set on the `EmailMessage`.

All of Djrill's other *Mandrill-specific options* can be used with templates.

Formatting Merge Data

If you're using dates, datetimes, Decimals, or anything other than strings and integers, you'll need to format them into strings for use as merge data:

```
product = Product.objects.get(123)   # A Django model
total_cost = Decimal('19.99')
ship_date = date(2015, 11, 18)

# Won't work -- you'll get "not JSON serializable" exceptions:
msg.global_merge_vars = {
    'PRODUCT': product,
    'TOTAL_COST': total_cost,
    'SHIP_DATE': ship_date
}

# Do something this instead:
msg.global_merge_vars = {
    'PRODUCT': product.name,   # assuming name is a CharField
    'TOTAL_COST': "%.2f" % total_cost,
    'SHIP_DATE': ship_date.strftime('%B %d, %Y')   # US-style "March 15, 2015"
}
```

These are just examples. You'll need to determine the best way to format your merge data as strings.

Although floats are allowed in merge vars, you'll generally want to format them into strings yourself to avoid surprises with floating-point precision.

Technically, Djrill will accept anything serializable by the Python json package – which means advanced template users can include dicts and lists as merge vars (for templates designed to handle objects and arrays). See the Python `json.JSONEncoder` docs for a list of allowable types.

How To Use Default Mandrill Subject and From fields

To use default Mandrill "subject" or "from" field from your template definition (overriding your EmailMessage and Django defaults), set the following attrs: *use_template_subject* and/or *use_template_from* on your `EmailMessage` object:

```
msg.use_template_subject = True
msg.use_template_from = True
msg.send()
```

use_template_subject

> If `True`, Djrill will omit the subject, and Mandrill will use the default subject from the template.
>
> New in version 1.1.

---

use_template_from

> If `True`, Djrill will omit the "from" field, and Mandrill will use the default "from" from the template.
>
> New in version 1.1.

1.4.2 Django Templates

To compose email using *Django* templates, you can use Django's `render_to_string()` template shortcut to build the body and html.

Example that builds an email from the templates `message_subject.txt`, `message_body.txt` and `message_body.html`:

```python
from django.core.mail import EmailMultiAlternatives
from django.template import Context
from django.template.loader import render_to_string

template_data = {
    'ORDERNO': "12345", 'TRACKINGNO': "1Z987"
}

plaintext_context = Context(autoescape=False)  # HTML escaping not appropriate in plaintext
subject = render_to_string("message_subject.txt", template_data, plaintext_context)
text_body = render_to_string("message_body.txt", template_data, plaintext_context)
html_body = render_to_string("message_body.html", template_data)

msg = EmailMultiAlternatives(subject=subject, from_email="store@example.com",
                             to=["customer@example.com"], body=text_body)
msg.attach_alternative(html_body, "text/html")
msg.send()
```

1.5 Mixing Email Backends

Since you are replacing Django's global `EMAIL_BACKEND`, by default Djrill will handle all outgoing mail, sending everything through Mandrill.

You can use Django mail's optional `connection` argument to send some mail through Mandrill and others through a different system.

This could be useful, for example, to deliver customer emails with Mandrill, but send admin emails directly through an SMTP server:

```python
from django.core.mail import send_mail, get_connection

# send_mail connection defaults to the settings EMAIL_BACKEND, which
# we've set to DjrillBackend. This will be sent using Mandrill:
send_mail("Thanks", "We sent your order", "sales@example.com", ["customer@example.com"])

# Get a connection to an SMTP backend, and send using that instead:
smtp_backend = get_connection('django.core.mail.backends.smtp.EmailBackend')
send_mail("Uh-Oh", "Need your attention", "admin@example.com", ["alert@example.com"],
    connection=smtp_backend)
```

You can supply a different connection to Django's `send_mail()` and `send_mass_mail()` helpers, and in the constructor for an `EmailMessage` or `EmailMultiAlternatives`.

(See the django.utils.log.AdminEmailHandler docs for more information on Django's admin error logging.)

1.6 Mandrill Webhooks and Inbound Email

Mandrill webhooks are used for notification about outbound messages (bounces, clicks, etc.), and also for delivering inbound email processed through Mandrill.

Djrill includes optional support for Mandrill's webhook notifications. If enabled, it will send a Django signal for each event in a webhook. Your code can connect to this signal for further processing.

New in version 0.5: Webhook support

> **Warning:** Webhook Security
> Webhooks are ordinary urls—they're wide open to the internet. You must take steps to secure webhooks, or anyone could submit random (or malicious) data to your app simply by invoking your webhook URL. For security:
> * Your webhook should only be accessible over SSL (https). (This is beyond the scope of Djrill.)
> * Your webhook must include a random, secret key, known only to your app and Mandrill. Djrill will verify calls to your webhook, and will reject calls without the correct key.
> * You can, optionally include the two settings `DJRILL_WEBHOOK_SIGNATURE_KEY` and `DJRILL_WEBHOOK_URL` to enforce webhook signature checking

1.6.1 Configuration

To enable Djrill webhook processing you need to create and set a webhook secret in your project settings, include the Djrill url routing, and then add the webhook in the Mandrill control panel.

1. In your project's `settings.py`, add a `DJRILL_WEBHOOK_SECRET`:

```
DJRILL_WEBHOOK_SECRET = "<create your own random secret>"
```

substituting a secret you've generated just for Mandrill webhooks. (Do *not* use your Mandrill API key or Django SECRET_KEY for this!)

An easy way to generate a random secret is to run the command below in a shell:

```
$ python -c "from django.utils import crypto; print crypto.get_random_string(16)"
```

2. In your base `urls.py`, add routing for the Djrill urls:

```
urlpatterns = patterns('',
    ...
    url(r'^djrill/', include(djrill.urls)),
)
```

3. Now you need to tell Mandrill about your webhook:

   * For receiving events on sent messages (e.g., bounces or clickthroughs), you'll do this in Mandrill's webhooks control panel.

   * For setting up inbound email through Mandrill, you'll add your webhook to Mandrill's inbound settings under "Routes" for your domain.

   * And if you want both, you'll need to add the webhook in both places.

   In all cases, the "Post to URL" is *https://yoursite.example.com*/djrill/webhook/?secret=*your-secret* substituting your app's own domain, and changing *your-secret* to the secret you created in step 1.

   (For sent-message webhooks, don't forget to tick the "Trigger on Events" checkboxes for the events you want to receive.)

Once you've completed these steps and your Django app is live on your site, you can use the Mandrill "Test" commands to verify your webhook configuration. Then see the next section for setting up Django signal handlers to process the webhooks.

Incidentally, you have some control over the webhook url. If you'd like to change the "djrill" prefix, that comes from the url config in step 2. And if you'd like to change the *name* of the "secret" query string parameter, you can set DJRILL_WEBHOOK_SECRET_NAME in your settings.py.

For extra security, Mandrill provides a signature in the request header X-Mandrill-Signature. If you want to verify this signature, you need to provide the settings DJRILL_WEBHOOK_SIGNATURE_KEY with the webhook-specific signature key that can be found in the Mandrill admin panel and DJRILL_WEBHOOK_URL where you should enter the exact URL, including that you entered in Mandrill when creating the webhook.

1.6.2 Webhook Notifications

Once you've enabled webhooks, Djrill will send a djrill.signals.webhook_event custom Django signal for each Mandrill event it receives. You can connect to this signal for further processing. Examples:

```python
from djrill.signals import webhook_event
from django.dispatch import receiver

@receiver(webhook_event)
def handle_bounce(sender, event_type, data, **kwargs):
    if event_type == 'hard_bounce' or event_type == 'soft_bounce':
        print "Message to %s bounced: %s" % (
            data['msg']['email'],
            data['msg']['bounce_description']
        )

@receiver(webhook_event)
def handle_inbound(sender, event_type, data, **kwargs):
    if event_type == 'inbound':
        print "Inbound message from %s: %s" % (
            data['msg']['from_email'],
            data['msg']['subject']
        )
```

Note that your webhook_event signal handlers will be called for all Mandrill webhook callbacks, so you should always check the event_type param as shown in the examples above to ensure you're processing the expected events.

Mandrill batches up multiple events into a single webhook call. Djrill will invoke your signal handler once for each event in the batch.

The available fields in the data param are described in Mandrill's documentation: sent-message webhooks and inbound webhooks.

1.7 Troubleshooting

Djrill throwing errors? Not sending what you want? Here are some tips...

1.7.1 Figuring Out What's Wrong

- **Check the error message:** Look for a Mandrill error message in your web browser or console (running Django in dev mode) or in your server error logs. As of v1.4, Djrill reports the detailed Mandrill error when something

goes wrong. And when the error is something like "invalid API key" or "invalid email address", that's probably 90% of what you'll need to know to solve the problem.

- **Check the Mandrill API logs:** The Mandrill dashboard includes an *incredibly-helpful* list of your recent API calls – and you can click into each one to see the full request and response. Check to see if the data you thought you were sending actually made it into the request, and if Mandrill has any complaints in the response.

- **Double-check common issues:**

    - Did you set your *MANDRILL_API_KEY* in settings.py?

    - Did you add 'djrill' to the list of INSTALLED_APPS in settings.py?

    - Are you using a valid from address? Django's default is "webmaster@localhost", which won't cut it. Either specify the from_email explicitly on every message you send through Djrill, or add DEFAULT_FROM_EMAIL to your settings.py.

- **Try it without Djrill:** Try switching your EMAIL_BACKEND setting to Django's File backend and then running your email-sending code again. If that causes errors, you'll know the issue is somewhere other than Djrill. And you can look through the EMAIL_FILE_PATH file contents afterward to see if you're generating the email you want.

1.7.2 Getting Help

If you've gone through the suggestions above and still aren't sure what's wrong, the Djrill community is happy to help. Djrill is supported and maintained by the people who use it – like you! (We're not Mandrill employees.)

You can ask in either of these places (but please pick only one per question!):

Ask on StackOverflow Tag your question with **both** Django and Mandrill to get our attention. Bonus: a lot of questions about Djrill are actually questions about Django itself, so by asking on StackOverflow you'll also get the benefit of the thousands of Django experts there.

Open a GitHub issue We do our best to answer questions in GitHub issues. And if you've found a Djrill bug, that's definitely the place to report it. (Or even fix it – see *Contributing*.)

Wherever you ask, it's always helpful to include the relevant portions of your code, the text of any error messages, and any exception stack traces in your question.

1.8 Contributing

Djrill is maintained by its users. Your contributions are encouraged!

The Djrill source code is on github. See AUTHORS.txt for a list of some of the people who have helped improve Djrill.

1.8.1 Bugs

You can report problems or request features in Djrill's github issue tracker.

We also have some *Troubleshooting* information that may be helpful.

1.8.2 Pull Requests

Pull requests are always welcome to fix bugs and improve support for Mandrill and Django features.

- Please include test cases.
- We try to follow the Django coding style (basically, PEP 8 with longer lines OK).
- By submitting a pull request, you're agreeing to release your changes under under the same BSD license as the rest of this project.

1.8.3 Testing

Djrill is tested on Travis against several combinations of Django and Python versions. (Full list in .travis.yml.)

Most of the included tests verify that Djrill constructs the expected Mandrill API calls, without actually calling Mandrill or sending any email. So these tests don't require a Mandrill API key, but they *do* require mock and six (`pip install mock six`).

To run the tests, either:

```
python -Wall setup.py test
```

or:

```
python -Wall runtests.py
```

If you set the environment variable `MANDRILL_TEST_API_KEY` to a valid Mandrill test API key, there are also a handful of integration tests which will run against the live Mandrill API. (Otherwise these live API tests are skipped.)

1.9 Release Notes

Djrill practices semantic versioning. Among other things, this means that minor updates (1.x to 1.y) should always be backwards-compatible, and breaking changes will always increment the major version number (1.x to 2.0).

1.9.1 Upcoming Changes in Djrill 2.0

Djrill 2.0 is under development and will include some breaking changes. Although the changes won't impact most Djrill users, the current version of Djrill (1.4) will try to warn you if you use things that will change. (Warnings appear in the console when running Django in debug mode.)

Djrill Admin site

Djrill 2.0 will remove the custom Djrill admin site. It duplicates information from Mandrill's dashboard, most Djrill users are unaware it exists, and it has caused problems tracking Django admin changes.

Drill 1.4 will report a DeprecationWarning when you try to load the `DjrillAdminSite`. You should remove it from your code.

Also, if you changed Django's `INSTALLED_APPS` setting to use `'django.contrib.admin.apps.SimpleAdminConfig'`, you may be able to switch that back to `'django.contrib.admin'` and let Django handle the admin.autodiscover() for you.

Dates in merge data and other attributes

Djrill automatically converts *send_at* date and datetime values to the ISO 8601 string format expected by the Mandrill API.

Unintentionally, it also converts dates used in other Mandrill message attributes (such as *merge_vars* or *metadata*) where it might not be expected (or appropriate).

Djrill 2.0 will remove this automatic date formatting, except for attributes that are inherently dates (currently only `send_at`).

To assist in detecting code relying on the (undocumented) current behavior, Djrill 1.4 will report a DeprecationWarning for date or datetime values used in any Mandrill message attributes other than `send_at`. See *Formatting Merge Data* for other options.

DjrillMessage class

The `DjrillMessage` class has not been needed since Djrill 0.2. You can simply set Djrill message attributes on any Django `EmailMessage` object. Djrill 1.4 will report a DeprecationWarning if you are still using DjrillMessage.

DjrillBackendHTTPError

The `DjrillBackendHTTPError` exception was replaced in Djrill 0.3 with *djrill.MandrillAPIError*. Djrill 1.4 will report a DeprecationWarning if you are still importing DjrillBackendHTTPError.

1.9.2 Change Log

Version 1.4:

- Django 1.8 support

- Support new Django 1.8 EmailMessage reply_to param. (Specifying a *Reply-To header* still works, with any version of Django, and will override the reply_to param if you use both.)

- Include Mandrill error response in str(MandrillAPIError), to make errors easier to understand.

- More-helpful exception when using a non-JSON-serializable type in merge_vars and other Djrill message attributes

- Deprecation warnings for upcoming 2.0 changes (see above)

Version 1.3:

- Use Mandrill secure https API endpoint (rather than http).

- Support *merge_language* option (for choosing between Handlebars and Mailchimp templates).

Version 1.2:

- Support Django 1.7; add testing on Python 3.3, 3.4, and PyPy

- Bug fixes

Version 1.1:

- Allow use of Mandrill template default "from" and "subject" fields, via *use_template_from* and *use_template_subject*.

- Fix `UnicodeEncodeError` with unicode attachments

Version 1.0:

- Global *MANDRILL_SUBACCOUNT* setting

Version 0.9:

- Better handling for "cc" and "bcc" recipients.

- Allow all extra message headers in send. (Mandrill has relaxed previous API restrictions on headers.)

Version 0.8:

- Expose *Mandrill Response* on sent messages

Version 0.7:

- Support for Mandrill send options *async*, *important*, *ip_pool*, *return_path_domain*, *send_at*, *subaccount*, and *view_content_link*

Version 0.6:

- Support for signed webhooks

Version 0.5:

- Support for incoming mail and other Mandrill webhooks

- Support for Mandrill send options *auto_html*, *tracking_domain* and *signing_domain*.

Version 0.4:

- Attachments with a Content-ID are now treated as *embedded images*

- New Mandrill *inline_css* option is supported

- Remove limitations on attachment types, to track Mandrill change

- Documentation is now available on djrill.readthedocs.org

Version 0.3:

- *Attachments* are now supported

- *Mandrill templates* are now supported

- A bcc address is now passed to Mandrill as bcc, rather than being lumped in with the "to" recipients. Multiple bcc recipients will now raise an exception, as Mandrill only allows one.

- Python 3 support (with Django 1.5)

- Exceptions should be more useful: *djrill.NotSupportedByMandrillError* replaces generic ValueError; *djrill.MandrillAPIError* replaces DjrillBackendHTTPError, and is now derived from requests.HTTPError. (New exceptions are backwards compatible with old ones for existing code.)

Version 0.2:

- MANDRILL_API_URL is no longer required in settings.py

- Earlier versions of Djrill required use of a DjrillMessage class to specify Mandrill-specific options. This is no longer needed – Mandrill options can now be set directly on a Django EmailMessage object or any subclass. (Existing code can continue to use DjrillMessage.)

Thanks

Thanks to the MailChimp team for asking us to build this nifty little app, and to all of Djrill's contributors. Also thanks to James Socol on Github for his django-adminplus library that got us off on the right foot for the custom admin views. Oh, and, of course, Kenneth Reitz for the awesome requests library.

A

async, 8
auto_html, 6
auto_text, 6

D

djrill.MandrillAPIError, 9
djrill.NotSupportedByMandrillError, 9

G

global_merge_vars, 7
google_analytics_campaign, 8
google_analytics_domains, 8

I

important, 6
inline_css, 6
ip_pool, 8

M

MANDRILL_API_KEY
    setting, 4
MANDRILL_SUBACCOUNT
    setting, 4
merge_language, 7
merge_vars, 7
metadata, 8

P

preserve_recipients, 7

R

recipient_metadata, 8
return_path_domain, 7

S

send_at, 8
setting
    MANDRILL_API_KEY, 4
    MANDRILL_SUBACCOUNT, 4

signing_domain, 7
subaccount, 7

T

tags, 7
track_clicks, 6
track_opens, 6
tracking_domain, 7

U

url_strip_qs, 7
use_template_from, 10
use_template_subject, 10

V

view_content_link, 7