

---

# **django-pymess Documentation**

*Release 1.4*

**Luboš Mátl**

**Jul 13, 2018**



---

## Contents

---

<b>1</b>	<b>Project Home</b>	<b>3</b>
<b>2</b>	<b>Documentation</b>	<b>5</b>
2.1	Content . . . . .	5



Django-pymess is a library for sending various type of messages like: SMS, Push notifications or e-mails



# CHAPTER 1

---

Project Home

---

<https://github.com/druids/django-pymess>



<https://django-pymess.readthedocs.org/en/latest>

## 2.1 Content

### 2.1.1 Installation

#### Using PIP

Django can use pip for installation:

```
$ pip install django-pymess
```

### 2.1.2 Configuration

After installation you must go through these steps:

#### Required Settings

The following variables have to be added to or edited in the project's `settings.py`:

For using `pymess` you just add `pymess` to `INSTALLED_APPS` variable:

```
INSTALLED_APPS = (  
    ...  
    'pymess',  
    ...  
)
```

### Setup

### SMS

#### **PYMESS\_SMS\_TEMPLATE\_MODEL**

If you want to use your own SMS template model you must set this setting with your custom SMS template model that extends `pymess.models.sms.AbstractSMSTemplate` otherwise `pymess.models.sms.SMSTemplate` is used.

#### **PYMESS\_SMS\_USE\_ACCENT**

Setting that sets if SMS will be sent with accent or not. Default value is `False`.

#### **PYMESS\_SMS\_LOG\_IDLE\_MESSAGES**

Setting that sets whether the delivery time is checked for messages. Default value is `True`.

#### **SMS\_SET\_ERROR\_TO\_IDLE\_MESSAGES**

Setting that sets if idle messages will be moved to the error state after defined time. Default value is `True`.

#### **PYMESS\_SMS\_IDLE\_MESSAGES\_TIMEOUT\_MINUTES**

If setting `PYMESS_SMS_LOG_IDLE_MESSAGES` is set to `True`, `PYMESS_SMS_IDLE_SENDING_MESSAGES_TIMEOUT_MINUTES` defines the number of minutes to send a warning that sms has not been sent. Default value is 10.

#### **PYMESS\_SMS\_DEFAULT\_PHONE\_CODE**

Country code that is set to the recipient if phone number doesn't contain another one.

#### **PYMESS\_SMS\_SENDER\_BACKEND**

Path to the SMS backend that will be used for sending SMS messages. Default value is `'pymess.backend.sms.dummy.DummySMSBackend'`.

#### **PYMESS\_SMS\_ATS\_CONFIG**

Configuration of `pymess.backend.sms.ats_sms_operator.ATSSMSBackend`.

#### **PYMESS\_SMS\_OPERATOR\_CONFIG**

Configuration of `pymess.backend.sms.sms_operator.SMSOperatorBackend`.

#### **PYMESS\_SMS\_SNS\_CONFIG**

Configuration of `pymess.backend.sms.sns.SNSSMSBackend`.

### E-MAIL

#### **PYMESS\_EMAIL\_TEMPLATE\_MODEL**

If you want to use your own E-MAIL template model you must set this setting with your custom e-mail template model that extends `pymess.models.email.AbstractEmailTemplate` otherwise is used `pymess.models.email.EmailTemplate`.

#### **PYMESS\_EMAIL\_SENDER\_BACKEND**

Path to the E-mail backend that will be used for sending e-mail messages. Default value is `'pymess.backend.emails.dummy.DummyEmailBackend'`.

#### **PYMESS\_EMAIL\_BATCH\_SENDING**

If you use standard SMTP service you should send e-mails in batches otherwise other SMTP providers could add your SMTP server to the black-list. With this setting you configure e-mail backend not to send e-mails directly but messages are only created in state "waiting". Finally e-mails should be sent with Django command `send_emails_batch`. Default value is `False`.

#### **PYMESS\_EMAIL\_BATCH\_SIZE**

Defines maximum number of e-mails that are sent with command `send_emails_batch`.

## PYMESS\_EMAIL\_MANDRILL

Configuration of `pymess.backend.email.mandrill.MandrillEmailBackend`.

## 2.1.3 SMS

SMS messages that are stored inside Django model class defined later, are sent via SMS backend. There are implemented several SMS backends, every backed uses differend SMS service like twillio or AWS SNS. For sending SMS message you can use function `pymess.backend.sms.send` or `pymess.backend.sms.send_template`.

`pymess.backend.sms.send(recipient, content, related_objects=None, tag=None, **sms_attrs)`

Function has two required parameters `recipient` which is a phone number of the receiver and `content`. Attribute `content` is a text message that will be sent inside the SMS body. If setting `PYMESS_SMS_USE_ACCENT` is set to `False`, accent in the content will be replaced by appropriate ascii characters. Attribute `related_objects` should contain a list of objects that you want to connect with the sent message (with generic relation). `tag` is string mark which is stored with the sent SMS message. The last non required parameter `**sms_kwargs` is extra data that will be stored inside SMS message model in field `extra_data`.

`pymess.backend.sms.send_template(recipient, slug, context_data, related_objects=None, tag=None)`

The second function is used for sending prepared templates that are stored inside template model (class that extends `pymess.models.sms.AbstractSMSTemplate`). The first parameter `recipient` is phone number of the receiver, `slug` is key of the template, `context_data` is a dictionary that contains context data for rendering SMS content from the template, `related_objects` should contains list of objects that you want to connect with the sent message and `tag` is string mark which is stored with the sent SMS message.

## Models

**class** `pymess.models.sms.OutputSMSMessage`

The model contains data of already sent SMS messages.

### **created\_at**

Django `DateTimeField`, contains date and time of creation.

### **changed\_at**

Django `DateTimeField`, contains date and time the of last change.

### **sent\_at**

Django `DateTimeField`, contains date and time of sending the SMS message.

### **recipient**

`CharField` that contains phone number of the receiver.

### **sender**

`CharField` that contains phone number of the sender. Field can be empty if backend doesn't provide sender number.

### **content**

`TextField`, contains content of the SMS message.

### **template\_slug**

If SMS was sent from the template, this attribute cointains key of the template.

### **template**

If SMS was sent from the template, this attribute contains foreign key of the template. The reason why there is `template_slug` and `template` fields is that a template instance can be removed and it is good to keep at least the key of the template.

### **state**

Field contains the current state of the message. Allowed states are:

- **WAITING** - SMS was not sent to the external service
- **UNKNOWN** - SMS was sent to the external service but its state is unknown
- **SENDING** - SMS was sent to the external service
- **SENT** - SMS was sent to the receiver
- **ERROR** - error was raised during sending of the SMS message
- **DEBUG** - SMS was not sent because system is in debug mode
- **DELIVERED** - SMS was delivered to the receiver

### **backend**

Field contains path to the SMS backend that was used for sending of the SMS message.

### **error**

If error was raised during sending of the SMS message this field contains text description of the error.

### **extra\_data**

Extra data stored with `JSONField`.

### **extra\_sender\_data**

Extra data related to the SMS backend stored with `JSONField`. Every SMS backend can have different extra data.

### **tag**

String tag that you can define during sending SMS message.

### **failed**

Returns `True` if SMS ended in `ERROR` state.

### **related\_objects**

Returns DB manager of `pymess.models.sms.OutputSMSRelatedObject` model that are related to the concrete SMS message.

### **class** `pymess.models.sms.OutputSMSRelatedObject`

Model for storing related objects that you can connect with the SMS message.

#### **created\_at**

Django `DateTimeField`, contains date and time of creation.

#### **changed\_at**

Django `DateTimeField`, contains date and time the of last change.

#### **output\_sms\_message**

Foreign key to the SMS message.

#### **content\_type**

Content type of the stored model (generic relation)

#### **object\_id\_int**

If a related objects has primary key in integer format the key is stored here. This field uses db index therefore filtering is much faster.

#### **object\_id**

Primary key of a related object stored in django `TextField`.

### **class** `pymess.models.sms.AbstractSMSTemplate`

Abstract class of SMS template which you can use to define your own SMS template model. Your model that extends this class is set inside setting `PYMESS_SMS_TEMPLATE_MODEL`:

```
PYMESS_SMS_TEMPLATE_MODEL = 'your_application.YourSMSTemplateModel'
```

**created\_at**

Django DateTimeField, contains date and time of creation.

**changed\_at**

Django DateTimeField, contains date and time the of last change.

**slug**

Key of the SMS template in the string format (Django slug).

**body**

Body of the SMS message. Final SMS content is rendered with Django template system by default.

**get\_body()**

Returns body of the model message. You can use it to update SMS body before rendering.

**render\_body(context\_data)**

Renders template stored inside `body` field to the message content. Standard Django template system is used by default.

**can\_send(recipient, context\_data)**

Returns by default `True` value. If you need to restrict sending SMS template for some reasons, you can override this method.

**send(recipient, context\_data, related\_objects=None, tag=None)**

Checks if message can be sent, renders message content and sends it via defined backend. Finally, the sent message is returned. If message cannot be sent, `None` is returned.

**class pymess.models.sms.SMSTemplate**

Default template model class that only inherits from `pymess.models.sms.AbstractSMSTemplate`

**Backends**

Backend is a class that is used for sending messages. Every backend must provide API defined by `pymess.backends.sms.SMSBackend` class. SMS backend is configured via `PYMESS_SMS_SENDER_BACKEND` (ex. `PYMESS_SMS_SENDER_BACKEND = 'pymess.backend.sms.sns.SNSSMSBackend'`). There are currently implemented following SMS backends:

**class pymess.backend.sms.dummy.DummySMSBackend**

Backend that can be used for testing. SMS is not sent, but is automatically set to the `DEBUG` state.

**class pymess.backend.sms.sns.SNSSMSBackend**

Backend that uses amazon SNS for sending messages (<https://aws.amazon.com/sns/>)

**class pymess.backend.sms.twilio.TwilioSMSBackend**

Backend that uses twilio service for sending SMS messages (<https://www.twilio.com/>)

**class pymess.backend.sms.ats\_sms\_operator.ATSSMSBackend**

Czech ATS SMS service is used for sending SMS messages. Service and backend supports checking if SMS was actually delivered. (<https://www.atspraha.cz/>)

Configuration of attributes according to ATS operator documentation:

```
PYMESS_ATS_SMS_CONFIG = {
    'URL': 'http://fik.atspraha.cz/gwfcgi/XMLServerWrapper.fcgi', # If you use
↪ default URL param, this doesn't need to be set
    'UNIQ_PREFIX': 'unique-id-prefix', # If you use SMS service for more
↪ applications you can define this prefix and it will be added to the message ID
```

(continues on next page)

(continued from previous page)

```

'USERNAME': 'username',
'PASSWORD': 'password',
'UNIQ_PREFIX': '',
'VALIDITY': 60,
'TEXTID': None,
'OPTID': ''
}

```

**class** pymess.backend.sms.sms\_operator.SMSOperatorBackend

Czech SMS operator service is used for sending SMS messages. Service and backend supports checking if SMS was actually delivered. (<https://www.sms-operator.cz/>)

Configuration of attributes according to SMS operator documentation:

```

PYMESS_SMS_OPERATOR_CONFIG = {
    'URL': 'https://www.sms-operator.cz/webservices/webservice.aspx', # If you
    ↪ use default URL param, this doesn't need to be set
    'UNIQ_PREFIX': 'unique-id-prefix', # If you uses SMS service for more
    ↪ applications you can define this prefix and it will be added to the message ID
    'USERNAME': 'username',
    'PASSWORD': 'password',
}

```

**Custom backend**

If you want to write your own Pymess SMS backend, you must create class that inherits from `pymess.backends.sms.SMSBackend`:

```

.. class pymess.backends.sms.SMSBackend

```

**publish\_message** (*message*)

This method should send SMS message (obtained from the input argument) and update its state. This method must be overridden in the custom backend.

**publish\_messages** (*messages*)

If your service that provides sending messages in batch, you can override the `publish_messages` method. Input argument is a list of messages. By default, `publish_message` method is used for sending and messages are send one by one.

**bulk\_check\_sms\_states** ()

If your service provides checking SMS state you can override this method and implement code that check if SMS messages were delivered.

**Commands**

Because some services provide checking if SMS messages were delivered, Pymess provides a command that calls backend method `bulk_check_sms_state`. You can use this command inside cron and periodically call it. But SMS backend and service must provide it (must have implemented method `bulk_check_sms_states`).

**2.1.4 E-mails**

Like SMS E-mail messages are stored inside Django model class and sent via backend. Again we provide more e-mail backends, every backend uses different e-mail service like Mandrill, AWS SNS or standard SMTP. For send-

ing e-mail message you can use function `pymess.backend.email.send` or `pymess.backend.email.send_template`.

```
pymess.backend.emails.send(sender, recipient, subject, content, sender_name=None,
                             related_objects=None, attachments=None, tag=None,
                             **email_kwargs)
```

Parameter `sender` define source e-mail address of the message, you can specify the name of the sender with optional parameter `sender_name`. `recipient` is destination e-mail address. Subject and HTML content of the e-mail message is defined with `subject` and `content` parameters. Attribute `related_objects` should contain a list of objects that you want to connect with the send message (with generic relation). Optional parameter `attachments` should contains list of files that will be sent with the e-mail in format `({file name}, {output stream with file content}, {content type})`. `tag` is string mark which is stored with the sent SMS message . The last non required parameter `**email_kwargs` is extra data that will be stored inside e-mail message model in field `extra_data`.

```
pymess.backend.emails.send_template(recipient, slug, context_data, related_objects=None,
                                     attachments=None, tag=None)
```

The second function is used for sending prepared templates that are stored inside template model (class that extends `pymess.models.sms.AbstractEmailTemplate`). The first parameter `recipient` is e-mail address of the receiver, `slug` is key of the template, `context_data` is a dictionary that contains context data for rendering e-mail content from the template, `related_objects` should contains list of objects that you want to connect with the send message, `attachments` should contains list of files that will be send with the e-mail and `tag` is string mark which is stored with the sent SMS message.

## Models

**class** `pymess.models.emails.EmailMessage`

The model contains data of already sent e-mail messages.

**created\_at**

Django DateTimeField, contains date and time of creation.

**changed\_at**

Django DateTimeField, contains date and time the of last change.

**sent\_at**

Django DateTimeField, contains date and time of sending the e-mail message.

**recipient**

EmailField that contains e-mail address of the receiver.

**sender**

EmailField that contains e-mail address of th sender.

**sender\_name**

CharField that contains readable/friendly sender name.

**subject**

TextField, contains subject of the e-mail message.

**content**

TextField, contains content of the e-mail message.

**template\_slug**

If e-mail was sent from the template, this attribute cointains key of the template.

**template**

If e-mail was sent from the template, this attribute contains foreign key of the template. The reason why there is `template_slug` and `template` fields is that a template instance can be removed and it is good to keep at least the key of the template.

### **state**

Contains the current state of the message. Allowed states are:

- **WAITING** - e-mail was not sent to the external service
- **SENDING** - e-mail was sent to the external service
- **SENT** - e-mail was sent to the receiver
- **ERROR** - error was raised during sending of the e-mail message
- **DEBUG** - e-mail was not sent because system is in debug mode

### **backend**

Field contains path to the e-mail backend that was used for sending of the SMS message.

### **error**

If error was raised during sending of the SMS message this field contains text description of the error.

### **extra\_data**

Extra data stored with `JSONField`.

### **extra\_sender\_data**

Extra data related to the e-mail backend stored with `JSONField`. Every SMS backend can have different extra data.

### **tag**

String tag that you can define during sending SMS message.

### **failed**

Returns `True` if SMS ended in `ERROR` state.

### **related\_objects**

Returns DB manager of `pymess.models.emails.EmailRelatedObject` model that are related to the concrete e-mail message.

## **class** `pymess.models.emails.EmailRelatedObject`

Model for storing related objects that you can connect with the e-mail message.

### **created\_at**

Django `DateTimeField`, contains date and time of creation.

### **changed\_at**

Django `DateTimeField`, contains date and time the of last change.

### **email\_message**

Foreign key to the e-mail message.

### **content\_type**

Content type of the stored model (generic relation)

### **object\_id\_int**

If a related objects have primary key in integer format the key is stored here. This field uses db index, therefore filtering is much faster.

### **object\_id**

Primary key of a related object stored in django `TextField`.

## **class** `pymess.models.emails.Attachment`

Django model that contains e-mail attachments.

### **created\_at**

Django `DateTimeField`, contains date and time of creation.

**changed\_at**

Django `DateTimeField`, contains date and time the of last change.

**email\_message**

Foreign key to the e-mail message.

**content\_type**

Django `CharField`, contains content type of the attachment.

**file**

Django `FileField`, contains file which was send to the recipient.

**class** `pymess.models.emails.AbstractEmailTemplate`

Abstract class of e-mail template which you can use to define your own e-mail template model. Your model that extends this class is set inside setting `PYMESS_EMAIL_TEMPLATE_MODEL`:

```
PYMESS_EMAIL_TEMPLATE_MODEL = 'your_application.YourEmailTemplateModel'
```

**created\_at**

Django `DateTimeField`, contains date and time of creation.

**changed\_at**

Django `DateTimeField`, contains date and time the of last change.

**slug**

Key of the e-mail template in the string format (Django slug).

**sender**

`EmailField` that contains e-mail address of the sender.

**sender\_name**

`CharField` that contains readable/friendly sender name.

**subject**

`TextField`, contains subject of the e-mail message. Final e-mail subject is rendered with Django template system by default.

**body**

Body of the e-mail message. Final e-mail content is rendered with Django template system by default.

**get\_body()**

Returns body of the model message. You can use it to update e-mail body before rendering.

**render\_body(*context\_data*)**

Renders template stored inside `body` field to the message content. Standard Django template system is used by default.

**get\_subject()**

Returns subject of the model message. You can use it to update e-mail subject before rendering.

**render\_subject(*context\_data*)**

Renders template stored inside `subject` field to the message content. Standard Django template system is used by default.

**can\_send(*recipient, context\_data*)**

Returns by default `True` value. If you need to restrict sending e-mail template for some reasons, you can override this method.

**send(*recipient, context\_data, related\_objects=None, tag=None, attachments=None*)**

Checks if message can be sent, renders message content and sends it via defined backend. Finally, the sent message is returned. If message cannot be sent, `None` is returned.

**class** `pymess.models.emails.EmailTemplate`  
Default template model class that only inherits from `pymess.models.emails.AbstractEmailTemplate`

### Backends

Backend is a class that is used for sending messages. Every backend must provide API defined by `pymess.backends.emails.EmailBackend` class. E-mail backend is configured via `PYMESS_EMAIL_SENDER_BACKEND` (ex. `PYMESS_EMAIL_SENDER_BACKEND = 'pymess.backends.emails.smtp.SMTPEmailBackend'`). There are currently implemented following e-mail backends:

**class** `pymess.backend.emails.dummy.DummyEmailBackend`  
Backend that can be used for testing. E-mail is not sent, but is automatically set to the `DEBUG` state.

**class** `pymess.backend.emails.smtp.SMTPEmailBackend`  
Backend that uses standard SMTP service for sending e-mails. Configuration of SMTP is same as Django configuration.

**class** `pymess.backend.emails.mandrill.MandrillEmailBackend`  
Backend that uses mandrill service for sending e-mail messages (<https://mandrillapp.com/api/docs/index.python.html>). For this purpose you must have installed mandrill library.

Configuration of attributes according to Mandrill operator documentation (the names of the configuration are the same):

```
PYMESS_EMAIL_MANDRILL_CONFIG = {
    'KEY': '', # Mandrill notification key
    'HEADERS': None,
    'TRACK_OPENS': False,
    'TRACK_CLICKS': False,
    'AUTO_TEXT': False,
    'INLINE_CSS': False,
    'URL_STRIP_QS': False,
    'PRESERVE_RECIPIENTS': False,
    'VIEW_CONTENT_LINK': True,
    'ASYNC': False,
}
```

### Custom backend

If you want to write your own Pymess e-mail backend, you must create class that inherits from `pymess.backends.emails.EmailBackend`:

```
.. class pymess.backends.sms.EmailBackend
```

**publish\_message** (*message*)

This method should send e-mail message (obtained from the input argument) and update its state. This method must be overridden in the custom backend.

### Commands

#### `send_emails_batch`

As mentioned e-mails can be sent in a batch with Django command `send_emails_batch`.

### `sync_emails`

Store e-mail body in a HTML file is better from code readability. Therefore this command updates e-mails body from HTML files store in directory. You can select the directory with command property `directory` or you can set directory with setting `PYMESS_EMAIL_HTML_DATA_DIRECTORY`. E-mails body in the directory is stored like HTML file named with e-mail slug and `html` as a suffix.

### `dump_emails`

E-mail body can be changed in the database therefore reverse operation to `sync_emails` can be done with this command. You must select directory where e-mails body in HTML format will be stored.



**B**

backend (pymess.models.emails.EmailMessage attribute), 12

backend (pymess.models.sms.OutputSMSMessage attribute), 8

body (pymess.models.emails.AbstractEmailTemplate attribute), 13

body (pymess.models.sms.AbstractSMSTemplate attribute), 9

bulk\_check\_sms\_states(), 10

**C**

can\_send() (pymess.models.emails.AbstractEmailTemplate method), 13

can\_send() (pymess.models.sms.AbstractSMSTemplate method), 9

changed\_at (pymess.models.emails.AbstractEmailTemplate attribute), 13

changed\_at (pymess.models.emails.Attachment attribute), 12

changed\_at (pymess.models.emails.EmailMessage attribute), 11

changed\_at (pymess.models.emails.EmailRelatedObject attribute), 12

changed\_at (pymess.models.sms.AbstractSMSTemplate attribute), 9

changed\_at (pymess.models.sms.OutputSMSMessage attribute), 7

changed\_at (pymess.models.sms.OutputSMSRelatedObject attribute), 8

content (pymess.models.emails.EmailMessage attribute), 11

content (pymess.models.sms.OutputSMSMessage attribute), 7

content\_type (pymess.models.emails.Attachment attribute), 13

content\_type (pymess.models.emails.EmailRelatedObject attribute), 12

content\_type (pymess.models.sms.OutputSMSRelatedObject

attribute), 8

created\_at (pymess.models.emails.AbstractEmailTemplate attribute), 13

created\_at (pymess.models.emails.Attachment attribute), 12

created\_at (pymess.models.emails.EmailMessage attribute), 11

created\_at (pymess.models.emails.EmailRelatedObject attribute), 12

created\_at (pymess.models.sms.AbstractSMSTemplate attribute), 9

created\_at (pymess.models.sms.OutputSMSMessage attribute), 7

created\_at (pymess.models.sms.OutputSMSRelatedObject attribute), 8

**E**

email\_message (pymess.models.emails.Attachment attribute), 13

email\_message (pymess.models.emails.EmailRelatedObject attribute), 12

error (pymess.models.emails.EmailMessage attribute), 12

error (pymess.models.sms.OutputSMSMessage attribute), 8

extra\_data (pymess.models.emails.EmailMessage attribute), 12

extra\_data (pymess.models.sms.OutputSMSMessage attribute), 8

extra\_sender\_data (pymess.models.emails.EmailMessage attribute), 12

extra\_sender\_data (pymess.models.sms.OutputSMSMessage attribute), 8

**F**

failed (pymess.models.emails.EmailMessage attribute), 12

failed (pymess.models.sms.OutputSMSMessage attribute), 8

file (pymess.models.emails.Attachment attribute), 13

## G

get\_body() (pymess.models.emails.AbstractEmailTemplate method), 13  
 get\_body() (pymess.models.sms.AbstractSMSTemplate method), 9  
 get\_subject() (pymess.models.emails.AbstractEmailTemplate method), 13

## O

object\_id (pymess.models.emails.EmailRelatedObject attribute), 12  
 object\_id (pymess.models.sms.OutputSMSRelatedObject attribute), 8  
 object\_id\_int (pymess.models.emails.EmailRelatedObject attribute), 12  
 object\_id\_int (pymess.models.sms.OutputSMSRelatedObject attribute), 8  
 output\_sms\_message (pymess.models.sms.OutputSMSRelatedObject attribute), 8

## P

publish\_message(), 10, 14  
 publish\_messages(), 10  
 pymess.backend.emails.dummy.DummyEmailBackend (built-in class), 14  
 pymess.backend.emails.mandrill.MandrillEmailBackend (built-in class), 14  
 pymess.backend.emails.send() (built-in function), 11  
 pymess.backend.emails.send\_template() (built-in function), 11  
 pymess.backend.emails.smtp.SMTPEmailBackend (built-in class), 14  
 pymess.backend.sms.ats\_sms\_operator.ATSSMSBackend (built-in class), 9  
 pymess.backend.sms.dummy.DummySMSBackend (built-in class), 9  
 pymess.backend.sms.send() (built-in function), 7  
 pymess.backend.sms.send\_template() (built-in function), 7  
 pymess.backend.sms.sms\_operator.SMSOperatorBackend (built-in class), 10  
 pymess.backend.sms.sns.SNSSMSBackend (built-in class), 9  
 pymess.backend.sms.twilio.TwilioSMSBackend (built-in class), 9  
 pymess.models.emails.AbstractEmailTemplate (built-in class), 13  
 pymess.models.emails.Attachment (built-in class), 12  
 pymess.models.emails.EmailMessage (built-in class), 11  
 pymess.models.emails.EmailRelatedObject (built-in class), 12  
 pymess.models.emails.EmailTemplate (built-in class), 13  
 pymess.models.sms.AbstractSMSTemplate (built-in class), 8

pymess.models.sms.OutputSMSMessage (built-in class), 7  
 pymess.models.sms.OutputSMSRelatedObject (built-in class), 8  
 pymess.models.sms.SMSTemplate (built-in class), 9  
 PYMESS\_EMAIL\_BATCH\_SENDING, 6  
 PYMESS\_EMAIL\_BATCH\_SIZE, 6  
 PYMESS\_EMAIL\_MANDRILL, 6  
 PYMESS\_EMAIL\_SENDER\_BACKEND, 6  
 PYMESS\_EMAIL\_TEMPLATE\_MODEL, 6  
 PYMESS\_SMS\_ATS\_CONFIG, 6  
 PYMESS\_SMS\_DEFAULT\_PHONE\_CODE, 6  
 PYMESS\_SMS\_IDLE\_MESSAGES\_TIMEOUT\_MINUTES, 6  
 PYMESS\_SMS\_LOG\_IDLE\_MESSAGES, 6  
 PYMESS\_SMS\_OPERATOR\_CONFIG, 6  
 PYMESS\_SMS\_SENDER\_BACKEND, 6  
 PYMESS\_SMS\_SNS\_CONFIG, 6  
 PYMESS\_SMS\_TEMPLATE\_MODEL, 6  
 PYMESS\_SMS\_USE\_ACCENT, 6

## R

recipient (pymess.models.emails.EmailMessage attribute), 11  
 recipient (pymess.models.sms.OutputSMSMessage attribute), 7  
 related\_objects (pymess.models.emails.EmailMessage attribute), 12  
 related\_objects (pymess.models.sms.OutputSMSMessage attribute), 8  
 render\_body() (pymess.models.emails.AbstractEmailTemplate method), 13  
 render\_body() (pymess.models.sms.AbstractSMSTemplate method), 9  
 render\_subject() (pymess.models.emails.AbstractEmailTemplate method), 13

## S

send() (pymess.models.emails.AbstractEmailTemplate method), 13  
 send() (pymess.models.sms.AbstractSMSTemplate method), 9  
 sender (pymess.models.emails.AbstractEmailTemplate attribute), 13  
 sender (pymess.models.emails.EmailMessage attribute), 11  
 sender (pymess.models.sms.OutputSMSMessage attribute), 7  
 sender\_name (pymess.models.emails.AbstractEmailTemplate attribute), 13  
 sender\_name (pymess.models.emails.EmailMessage attribute), 11  
 sent\_at (pymess.models.emails.EmailMessage attribute), 11

sent\_at (pymess.models.sms.OutputSMSMessage attribute), 7

slug (pymess.models.emails.AbstractEmailTemplate attribute), 13

slug (pymess.models.sms.AbstractSMSTemplate attribute), 9

SMS\_SET\_ERROR\_TO\_IDLE\_MESSAGES, 6

state (pymess.models.emails.EmailMessage attribute), 12

state (pymess.models.sms.OutputSMSMessage attribute), 8

subject (pymess.models.emails.AbstractEmailTemplate attribute), 13

subject (pymess.models.emails.EmailMessage attribute), 11

## T

tag (pymess.models.emails.EmailMessage attribute), 12

tag (pymess.models.sms.OutputSMSMessage attribute), 8

template (pymess.models.emails.EmailMessage attribute), 11

template (pymess.models.sms.OutputSMSMessage attribute), 7

template\_slug (pymess.models.emails.EmailMessage attribute), 11

template\_slug (pymess.models.sms.OutputSMSMessage attribute), 7