Django Webix Sender Documentation

Release 1.0.0

MPA Solutions

Jan 22, 2021

Contents

1 Contents **1**

1.1 Django Webix Sender 1
1.2 Quick Start 2
1.3 Customization 8
1.4 Credits 10
1.5 Class Reference 11
1.6 Change Log 23

2 Indices and tables **27**

Python Module Index **29**

Index **31**

CHAPTER 1

Contents

1.1 Django Webix Sender

1.1.1 Documentation

The full documentation is at https://django-webix-sender.readthedocs.io.

1.1.2 Quickstart

Install Django Webix Sender:

```
$ pip install django-webix-sender
```

Add `django-webix-sender` to your `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'django_webix_sender',
    # ...
]
```

Add `django-webix-sender` URLconf to your project `urls.py` file

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

urlpatterns = [
    # ...
    url(r'^django-webix-sender/', include('django_webix_sender.urls')),
    # ...
]
```

Warning: This package requires a project with `django-webix` setted up.

Warning: This package requires 'django.contrib.humanize' in `INSTALLED_APPS`

1.1.3 Running Tests

Does the code actually work?

```
$ source <YOURVIRTUALENV>/bin/activate
$ (myenv) $ pip install tox
$ (myenv) $ tox
```

1.1.4 Contributors

Here is a list of Django-Webix's contributors.

1.2 Quick Start

1.2.1 Install

`django-webix-sender` is available on `https://pypi.python.org/pypi/django-webix-sender/` install it simply with:

```
$ pip install django-webix-sender
```

1.2.2 Configure

Settings

Add `django_webix_sender` to your `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'django_webix_sender',
    # ...
]
```

Add `django-webix-sender` URLconf to your project `urls.py` file

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

urlpatterns = [
    # ...
    url(r'^django-webix-sender/', include('django_webix_sender.urls')),
    # ...
]
```

Warning: This package requires a project with `django-webix` setted up.

Warning: This package requires 'django.contrib.humanize' in `INSTALLED_APPS`

1.2.3 Usage

Settings

Create the models (e.g. <app_name>/models.py)

```
from django.utils.translation import gettext_lazy as _
from django_webix_sender.send_methods.telegram.handlers import start, check_user

WEBIX_SENDER = {
    'send_methods': [
        {
            'method': 'skebby',
            'verbose_name': _('Send sms'),
            'function': 'django_webix_sender.send_methods.skebby.send',
            'show_in_list': True,
            'show_in_chat': False,
            'config': {
                'region': "IT",
                'method': SkebbyMessageType.GP,
                'username': 'username',
                'password': '********',
                'sender_string': 'Sender',
            }
        },
        {
            'method': 'email',
            'verbose_name': _('Send email'),
            'function': 'django_webix_sender.send_methods.email.send',
            'show_in_list': True,
            'show_in_chat': False,
            'config': {
                'from_email': 'noreply@email.com'
            }
        },
        {
            'method': 'telegram',
            'verbose_name': _('Send telegram'),
```

(continues on next page)

(continued from previous page)

```
            'function': 'django_webix_sender.send_methods.telegram.send',
            'show_in_list': False,
            'show_in_chat': True,
            'config': {
                "bot_token": "**********:**********",
                "webhooks": [
                    "https://mysite.com/django-webix-sender/telegram/webhook/"
                ],
                'commands': [
                    BotCommand("start", "Start info"),
                ],
                'handlers': [
                    {"handler": MessageHandler(Filters.all, check_user), "group": -1},
↪   # Check enabled users
                    CommandHandler("start", start),  # Example
                ]
            }
        },
        {
            'method': 'storage',
            'verbose_name': _('Store online'),
            'function': 'django_webix_sender.send_methods.storage.send',
            'show_in_list': True,
            'show_in_chat': False,
        },
    ],
    'initial_send_methods': [
        {
            'method': 'storage',
            'function': 'django_webix_sender.send_methods.storage.send',
        },
        {
            'method': 'telegram',
            'function': 'django_webix_sender.send_methods.telegram.send',
        },
    ],
    'attachments': {
        'model': 'django_webix_sender.MessageAttachment',
        'upload_folder': 'sender/',
        'save_function': 'django_webix_sender.models.save_attachments'
    },
    'typology_model': {
        'enabled': True,
        'required': False
    },
    'recipients': [
        {
            'model': 'django_webix_sender.Customer',
            'datatable_fields': ['user', 'name', 'sms', 'email', 'telegram'],
            'collapsed': False
        },
        {
            'model': 'django_webix_sender.ExternalSubject',
            'datatable_fields': ['user', 'name', 'sms', 'email', 'telegram'],
            'collapsed': True
        },
    ],
```

(continues on next page)

(continued from previous page)

```
    'groups_can_send': ["Admin"],
    'extra': {
        'session': ['year']
    },
    'invoices_period': 'bimestrial'
}
```

WEBIX_SENDER['send_methods']

Defines the allowed send methods.

There are four allowed methods type:

- `skebby`
- `email`
- `telegram`
- `storage`

The methods already implemented in this package are:

- `django_webix_sender.send_methods.email.send`

  The default Django email sender.

  ```
  {
      'method': 'email',
      'verbose_name': _('Send email'),
      'function': 'django_webix_sender.send_methods.email.send',
      'show_in_list': True,
      'show_in_chat': False,
      'config': {
          'from_email': 'noreply@email.com'
      }
  }
  ```

- `django_webix_sender.send_methods.skebby.send`

  Skebby sms APIs.

  ```
  {
      'method': 'skebby',
      'verbose_name': _('Send sms with Skebby'),
      'function': 'django_webix_sender.send_methods.skebby.send',
      'show_in_list': True,
      'show_in_chat': False,
      'config': {
          'region': "IT",
          'method': SkebbyMessageType.GP,
          'username': 'username',
          'password': '********',
          'sender_string': 'Sender',
      }
  }
  ```

- `django_webix_sender.send_methods.telegram.send`

  Telegram APIs.

```
{
    'method': 'telegram',
    'verbose_name': _('Send with Telegram'),
    'function': 'django_webix_sender.send_methods.telegram.send',
    'show_in_list': False,
    'show_in_chat': True,
    'config': {
        "bot_token": "**********:**********",
        "webhooks": [
            "https://mysite.com/django-webix-sender/telegram/webhook/"
        ],
        'commands': [
            BotCommand("start", "Start info"),
        ],
        'handlers': [
            {"handler": MessageHandler(Filters.all, check_user), "group
↪": -1},  # Check enabled users
            CommandHandler("start", start),  # Example
        ]
    }
}
```

- `django_webix_sender.send_methods.storage.send`

  Storage method

```
{
    'method': 'storage',
    'verbose_name': _('Store online'),
    'function': 'django_webix_sender.send_methods.storage.send',
    'show_in_list': True,
    'show_in_chat': False,
}
```

`WEBIX_SENDER['initial_send_methods']`

Optional: Defines the default send methods in the form.

```
[
    {
        'method': 'storage',
        'function': 'django_webix_sender.send_methods.storage.send',
    },
    {
        'method': 'telegram',
        'function': 'django_webix_sender.send_methods.telegram.send',
    },
]
```

`WEBIX_SENDER['attachments']`

Defines the attachments model and the method to store files.

```
{
    'model': 'django_webix_sender.MessageAttachment',
    'upload_folder': 'sender/',
    'save_function': 'django_webix_sender.models.save_attachments'
}
```

`WEBIX_SENDER['typology_model']`

Defines if the message typology are enabled.

```
{
    'enabled': True,
    'required': False
}
```

WEBIX_SENDER['recipients']
Defines the models to show as a list of recipients.

```
{
    'model': 'django_webix_sender.Customer',
    'datatable_fields': ['user', 'name', 'sms', 'email', 'telegram'],
    'collapsed': True
}
```

WEBIX_SENDER['groups_can_send']
Optional: Defines the group names that can send messages.

```
["Admin"]
```

WEBIX_SENDER['extra']
Optional: Defines the data to add to message extra json field. You can define variable names in the session.

```
{
    'session': ['year']
}
```

WEBIX_SENDER['invoices_period']
Optional: Defines the periods to divide the invoices.

The available periods are:

- `monthly`
- `bimestrial`
- `quarter`
- `half-yearly`
- `yearly`

Warning: You can add `get_sender` method to the user class to indicate string to be stored in the message record

```
def _get_sender(self):
    return self.get_full_name()

User.get_sender = _get_sender
```

Base Template

Create a base html template (e.g. <app_name>/templates/base.html)

```
{% load i18n %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    {% include "django_webix/static_meta.html" %}
</head>
<body>
</body>

<script type="text/javascript" charset="utf-8">
    webix.ready(function () {
        webix.ui({
            id: 'content_right',
            rows: []
        });

        webix.extend($$('content_right'), webix.OverlayBox);

        load_js('{% url 'django_webix_sender.list' %}');
    });
</script>
</html>
```

1.3 Customization

1.3.1 Recipient class

Create a subclass of `DjangoWebixSender` and define `get_sms`, `get_telegram`, `get_email`, `get_sms_related`, `get_telegram_related` and `get_email_related` properties, if you use that send method.

It's important to define also `get_email_related`, `get_sms_fieldpath`, `get_email_fieldpath` and `get_telegram_fieldpath` classmethods.

Optionally you can define also `get_select_related` and `get_prefetch_related` to optimize queries.

There is also `get_filters_viewers` function to define which qset to apply to the recipients models to indicates their visibility.

```
class Recipients(DjangoWebixSender):
    name = models.CharField(max_length=255, verbose_name=_('Name'))
    sms = models.CharField(max_length=32, blank=True, null=True, verbose_name=_('Sms
↪'))
    telegram = models.CharField(max_length=32, blank=True, null=True, verbose_name=_(
↪'Telegram'))
    email = models.EmailField(max_length=255, blank=True, null=True, verbose_name=_(
↪'Email'))
    parent = models.ForeignKey('self', blank=True, null=True, verbose_name=_('Parent
↪'))
```

(continues on next page)

(continued from previous page)

```
    @property
    def get_sms(self):
        return self.sms

    @property
    def get_telegram(self):
        return self.telegram

    @property
    def get_email(self):
        return self.email

    @property
    def get_sms_related(self):
        return self.parent_set.all()

    @property
    def get_telegram_related(self):
        return self.parent_set.all()

    @property
    def get_email_related(self):
        return self.parent_set.all()

    @staticmethod
    def get_sms_fieldpath() -> str:
        return "sms"

    @staticmethod
    def get_email_fieldpath() -> str:
        return "email"

    @staticmethod
    def get_telegram_fieldpath() -> str:
        return "telegram"

    @classmethod
    def get_filters_viewers(cls, user, *args, **kwargs) -> Q:
        if user is None:
            return Q(pk__isnull=True)  # Fake filter, empty queryset
        if user.is_anonymous:
            return Q(pk__isnull=True)  # Fake filter, empty queryset
        if not user.is_superuser:
            return Q(user=user)
        return Q()  # Non filters

    @classmethod
    def get_representation(cls) -> F:
        return F('name')
```

Warning: You need to define `get_filters_viewers` staticmethod to the recipient models to filter recipients that can be seen by passed user. This method must returns QSet object.

```
@classmethod
def get_filters_viewers(cls, user, *args, **kwargs) -> Q:
    if user is None:
        return Q(pk__isnull=True)  # Fake filter, empty queryset
    if user.is_anonymous:
        return Q(pk__isnull=True)  # Fake filter, empty queryset
    if not user.is_superuser:
        return Q(user=user)
    return Q()  # Non filters
```

1.3.2 Send method

```
def send_sms(recipients, body, message_sent):

    # ...
    # API gateway sms send
    # ...

    for recipient, recipient_address in recipients['valids']:
        MessageRecipient.objects.create(
            message_sent=message_sent,
            recipient=recipient,
            sent_number=1,
            status='success',
            recipient_address=recipient_address
        )
    for recipient, recipient_address in recipients['invalids']:
        pass  # You must create MessageRecipient instance
    for recipient, recipient_address in recipients['duplicates']:
        pass  # You must create MessageRecipient instance
    return message_sent
```

1.4 Credits

This package was developed by MPA Solutions

1.4.1 Development Lead

- Alessio Bazzanella <bazzanella@mpasol.it>
- Alessandro Regolini <regolini@mpasol.it>

1.4.2 Contributors

None yet. Why not be the first?

1.5 Class Reference

1.5.1 Models

django_webix_sender.models.**save_attachments**(*files: Dict[str, Any], *args, **kwargs*)
Save attachments function

Parameters

- **files** – a dict with attachments
- **args** – Optional arguments
- **kwargs** – optional keyword arguments

Returns list of MessageAttachment instances

class django_webix_sender.models.**DjangoWebixSender**(**args*, ***kwargs*)
Abstract model with basic configuration

get_email
Get email address

Returns email address

static get_email_fieldpath() → str
Get email field name (or complete path with fks)

Returns string with email fieldname

get_email_related
Get all email related recipients to this instance

Returns list of instances of related recipients

classmethod get_filters_viewers(*user*, **args*, ***kwargs*) → django.db.models.query_utils.Q
Filters recipients that can be seen by this user

Parameters

- **user** – user instance
- **args** – Optional arguments
- **kwargs** – optional keyword arguments

Returns Q object with filters

classmethod get_prefetch_related() → List[str]
Related field to optimize django queries

Returns list of prefetch related fields

classmethod get_representation() → django.db.models.expressions.F
Get field to create a sql rapresentation

Returns F object

classmethod get_select_related() → List[str]
Related field to optimize django queries

Returns list of select related fields

get_sms
Get sms number

Returns sms number

static get_sms_fieldpath() → str
Get sms field name (or complete path with fks)

Returns string with sms fieldname

get_sms_related
Get all sms related recipients to this instance

Returns list of instances of related recipients

get_telegram
Get telegram id

Returns telegram id

static get_telegram_fieldpath() → str
Get telegram field name (or complete path with fks)

Returns string with telegram fieldname

get_telegram_related
Get all email related recipients to this instance

Returns list of instances of related recipients

class django_webix_sender.models.**Customer**(**args*, ***kwargs*)
Customer model

exception DoesNotExist

exception MultipleObjectsReturned

get_email
Get email address

Returns email address

static get_email_fieldpath() → str
Get email field name (or complete path with fks)

Returns string with email fieldname

classmethod get_filters_viewers(*user*, **args*, ***kwargs*) → django.db.models.query_utils.Q
Filters recipients that can be seen by this user

Parameters

- **user** – user instance
- **args** – Optional arguments
- **kwargs** – optional keyword arguments

Returns Q object with filters

classmethod get_representation() → django.db.models.expressions.F
Get field to create a sql rapresentation

Returns F object

get_sms
Get sms number

Returns sms number

static get_sms_fieldpath() → str
Get sms field name (or complete path with fks)

Returns string with sms fieldname

get_telegram
Get telegram id

Returns telegram id

static get_telegram_fieldpath() → str
Get telegram field name (or complete path with fks)

Returns string with telegram fieldname

class django_webix_sender.models.**CustomerTypology**(**args*, ***kwargs*)
Customer typology model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**ExternalSubject**(**args*, ***kwargs*)
External subject model

exception DoesNotExist

exception MultipleObjectsReturned

get_email
Get email address

Returns email address

static get_email_fieldpath() → str
Get email field name (or complete path with fks)

Returns string with email fieldname

classmethod get_filters_viewers(*user*, **args*, ***kwargs*) → django.db.models.query_utils.Q
Filters recipients that can be seen by this user

Parameters

- **user** – user instance
- **args** – Optional arguments
- **kwargs** – optional keyword arguments

Returns Q object with filters

classmethod get_representation() → django.db.models.expressions.F
Get field to create a sql rapresentation

Returns F object

get_sms
Get sms number

Returns sms number

static get_sms_fieldpath() → str
Get sms field name (or complete path with fks)

Returns string with sms fieldname

get_telegram
Get telegram id

Returns telegram id

static get_telegram_fieldpath() → str
Get telegram field name (or complete path with fks)

Returns string with telegram fieldname

class django_webix_sender.models.**ExternalSubjectTypology**(**args*, ***kwargs*)
External subject typology model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**MessageAttachment**(**args*, ***kwargs*)
Message attachments model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**MessageTypology**(**args*, ***kwargs*)
Message typology model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**MessageSent**(**args*, ***kwargs*)
Message sent model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**MessageRecipient**(**args*, ***kwargs*)
Message recipient model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**MessageUserRead**(**args*, ***kwargs*)
Message readed by user model

exception DoesNotExist

exception MultipleObjectsReturned

class django_webix_sender.models.**TelegramPersistence**(**args*, ***kwargs*)
Telegram persistence model

exception DoesNotExist

exception MultipleObjectsReturned

1.5.2 Template Tags

django_webix_sender.templatetags.field_type.**field_type**(*context*, *model*, *field_name*)
Returns model field type

django_webix_sender.templatetags.send_methods_utils.**is_chat_available**(*context*)
Returns boolean to indicate if there are chats configured

django_webix_sender.templatetags.send_methods_utils.**is_list_available**(*context*)
Returns boolean to indicate if there are lists configured

django_webix_sender.templatetags.send_methods_utils.**user_can_send**(*context*)
Returns boolean to indicate if user has send permission

django_webix_sender.templatetags.verbose_name.**get_verbose_field_name**(*context*, *model*, *field_name*)
Returns verbose_name for a field.

1.5.3 Views

1.5.4 Utils

django_webix_sender.utils.**my_import**(*name: str*) → callable
Load a function from a string

Parameters **name** – function path name (e.g. django_webix_sender.send_methods.email.send_utils)

Returns callable

django_webix_sender.utils.**send_mixin**(*send_method: str, typology: Optional[int], subject: str, body: str, recipients: Dict[str, List[int]], presend: Optional[Any], **kwargs*) → Tuple[Dict[str, Any], int]
Function to send the message

Parameters

- **send_method** – <skebby|email|telegram|storage>.<function> (eg. "skebby.django_webix_sender.send_methods.email.send_utils")
- **typology** – MessageTypology ID
- **subject** – Subject of message
- **body** – Body of message (email, skebby, telegram or storage)
- **recipients** – Dict {'<app_label>.<model>': [<id>, <id>]}
- **presend** – None: verify before the send; Otherwise: send the message
- **kwargs** – *user* and *files* (default: user=None, files={})

Returns Tuple[Dict, Code]

1.5.5 Send Methods

Email

Send Utils

django_webix_sender.send_methods.email.send_utils.**send**(*recipients: Dict[str, List[int]], subject: str, body: str, message_sent*)
Send email

Parameters

- **recipients** – Dict {'<app_label>.<model>': [<id>, <id>]}

- **subject** – Subject of email
- **body** – Body of message
- **message_sent** – MessageSent instance

Returns MessageSent instance

Skebby

Enums

class django_webix_sender.send_methods.skebby.enums.**SkebbyBoolean**
An enumeration.

class django_webix_sender.send_methods.skebby.enums.**SkebbyEncoding**
An enumeration.

class django_webix_sender.send_methods.skebby.enums.**SkebbyMessageType**
An enumeration.

Exceptions

exception django_webix_sender.send_methods.skebby.exceptions.**SkebbyException**

Gateway

class django_webix_sender.send_methods.skebby.gateway.**Skebby**
https://developers.skebby.it

class Authentication
Authentication methods

The following are the two available methods to authenticate a user, given a username and a password (registration required): - Using a temporary session key, which expires after a certain amount of time has passed with no performed API calls with that key. - Using an authentication token, which does not expire, except when an account is deactivated or suspended. In both cases, the returned user_key, as well as the session key or the token, are required to be provided in the HTTP request headers in order to perform any API call after the login.

session_key(*username*, *password*)
Authenticate using a session key

The login with session key API lets you authenticate by using your username and password, and returns a token to be used for authenticating the next API calls. The following HTTP headers should be provided after the login: - user_key:USER_KEY - Session_key:SESSION_KEY Where USER_KEY and SESSION_KEY are the values returned by the login API.

user_token(*username*, *password*)
Authenticate using a user token

The login with token API lets you authenticate by using your username and password, and returns a token to be used for authenticating the next API calls. The following HTTP headers should be provided after the login: - user_key:USER_KEY - Access_token:ACCESS_TOKEN Where USER_KEY and ACCESS_TOKEN are the values returned by the login API.

class Contacts(*authentication*)
Contacts API

This part of the API is used to manage contacts.

add_contact(*email*, *phone_number*, *name=''*, *surname=''*, *gender=''*, *fax=''*, *address=''*, *city=''*, *province=''*, *birthdate=''*, *promotiondate=''*, *rememberdate=''*, *zip_code=''*, *group_ids=None*, *custom1=''*, *custom2=''*, *custom3=''*, *custom4=''*, *custom5=''*, *custom6=''*, *custom7=''*, *custom8=''*, *custom9=''*, *custom10=''*)
Add a contact

Add a contact to the user's addressbook.

class ContactsGroups(*authentication*)
Contacts groups API

This section describes how groups of contacts are created, updated and deleted. SMS messages can be directly sent to groups of contacts.

class LandingPages(*authentication*)
Landing pages API

This is the part of the API that is concerned with the landing pages service.

class ReceivedSMS(*authentication*)
Received SMS API

This API allows to query the received SMS messages on the owned SIMs.

class SmsBlacklist(*authentication*)
SMS Blacklist / Stop SMS

This is the part of the API that allow to insert and to retrieve the list of SMS blacklist / Stop SMS. The SMS blacklist contains the phone numbers to which you don't want to send any SMS. If the Stop SMS Service is active, any person who receive an SMS can add their phone number to the blacklist.

class SmsHistory(*authentication*)
SMS History API

This API is used to retrieve the SMS messages sending history.

get_sent_rich_sms_statistics(*order_id*)
Get sent Rich SMS statistics

After sending an sms campaign via API containing a Rich sms shortened link, this api returns that sms link's opening statistics. Keep in mind that you need to use a unique order_id when sending and store it to request in the near future.

get_sent_sms_history(*date_from*, *date_to='now'*, *timezone=None*, *page_number=1*, *page_size=10*)
Get sent SMS history

Returns the user's SMS messages history

get_sent_sms_to_recipient(*recipient*, *date_from*, *date_to='now'*, *timezone=None*, *page_number=1*, *page_size=10*)
Get sent SMS to a recipient

Returns the user's SMS messages history for the specified recipient

class SmsSend(*authentication*)
SMS send API

This is the part of the API that allows to send SMS messages, to single recipients, saved contacts or groups of contacts.

get_sms_state(*order_id*)
Get SMS message state

Get informations on the SMS delivery status of the given order_id.

send_parametric_sms(*message_type: django_webix_sender.send_methods.skebby.enums.SkebbyMessageType*, *message*, *recipient*, *sender=''*, *scheduled_delivery_time=None*, *scheduled_delivery_timezone=None*, *order_id=None*, *return_credits=<SkebbyBoolean.FALSE: 'false'>*, *return_remaining=<SkebbyBoolean.FALSE: 'false'>*, *allow_invalid_recipients=<SkebbyBoolean.FALSE: 'false'>*, *encoding=<SkebbyEncoding.GSM: 'gsm'>*, *id_landing=None*, *campaign_name=None*, *max_fragments=7*, *truncate=<SkebbyBoolean.TRUE: 'true'>*, *richsms_url=None*, *richsms_mode=None*)
Send a parametric SMS message

Sends a parametric SMS message to a given list of recipients. With this API it is possible to put placeholders in the message body, and then, for each recipient, specify the values that will replace the placeholders in the message body, for that particular recipient message. Placeholders are in the form ${ParameterName}

Landing Pages URLs It is possible to include a link to a published Landing Page by specifying the id_landing parameter and by adding the following placeholder in the message body: %PAGES-LINK_____%.

Landing pages must be first created and published in your user panel, since you will need id_landing to send it. A list of published landing pages can be retrieved by using the Landing Pages APIs

SMS Link Analytics When including URLs in the message, it may be convenient to use our SMS Link Analytics short URLs service to limit the number of characters used in the SMS message and having statistic on clic. Our API can automatically generate a short link starting from a long one, and add it in the message. To use this feature, use the %RICHURL_____% placeholder in the message body and set the parameter rich_mode with one of following values: DIRECT_URL: in this case you must add the parameter richsms_url with the url that you want to be shortened RECIPIENT: in this case the url must be set in the url property for each recipient in recipients parameter. You could omit richsms_mode if you specify both the richsms_url params and %RICHURL_____% placeholder.

Aliases can be used only with high-quality message types.

send_sms(*message_type: django_webix_sender.send_methods.skebby.enums.SkebbyMessageType*, *message*, *recipient*, *sender=''*, *scheduled_delivery_time=None*, *scheduled_delivery_timezone=None*, *order_id=None*, *return_credits=<SkebbyBoolean.FALSE: 'false'>*, *return_remaining=<SkebbyBoolean.FALSE: 'false'>*, *allow_invalid_recipients=<SkebbyBoolean.FALSE: 'false'>*, *encoding=<SkebbyEncoding.GSM: 'gsm'>*, *id_landing=None*, *campaign_name=None*, *max_fragments=7*, *truncate=<SkebbyBoolean.TRUE: 'true'>*, *richsms_url=None*)
Send an SMS message

Sends an SMS message to a given list of recipients.

Landing Pages URLs It is possible to include a link to a published Landing Page by specifying the id_landing parameter and by adding the following placeholder in the message body: %PAGES-LINK_____%.

Landing pages must be first created and published in your user panel, since you will need id_landing to send it. A list of published landing pages can be retrieved by using the Landing Pages APIs

SMS Link Analytics When including URLs in the message, it may be convenient to use our SMS Link Analytics short URLs service to limit the number of characters used in the SMS message and having statistic on clic. Our API can automatically generate a short link starting from a long one, and add it in the message. To use this feature, use the %RICHURL_____% placeholder in the message body, that will be replaced with the generated short link, and the respective richsms_url parameter, that should be set to a valid URL.

Sender Alias Alphanumeric aliases are required to be registered first, and need to be approved both from Us and AGCOM. Aliases can be used only with high-quality message types.

class Subaccount(*authentication*)
Subaccount API

If enabled as a superaccount, the user can create subaccounts that can be assigned to third-parties. Superaccounts may or may not share credits with their subaccounts.

class TPOA(*authentication*)
TPOA API

The TPOA (Transmission Path Originating Address) API is used to deal with TPOA entries (i.e. "SMS sender aliases") of the user.

class TwoFactorAuthentication(*authentication*)
Two Factor Authentication API

This is the part of the API that provides the Two Factor Authentication. The flow of 2FA is: 1. The user specifies their number in your App. 2. Your app sends a 2FA request via API. 3. The platform sends a text message to the specified recipient. 4. User receives the PIN via text message. 5. User enters the PIN in your App. 6. Your app sends the 2FA verify via API and receives the authorization or an invalid pin error.

The text message is sent with the highest quality on a preferred route, to guarantee a quick delivery.

class User(*authentication*)
User API

The following are utility functions regarding the Authenticated User (e.g. the user status, password reset, etc)

dashboard()
Dashboard

API used to retrieve the dashboard URL of the authenticated user

reset_password(*password*)
Reset password

Changes the authenticated user's password

user_status(*get_money=<SkebbyBoolean.FALSE: 'false'>*, *type_aliases=<SkebbyBoolean.FALSE: 'false'>*)
User status

Used to retrieve the credits and other information of the user identified by the id.

verify_session()
Verify session

Checks whether the user session is still active and valid (without renewal).

Send Utils

django_webix_sender.send_methods.skebby.send_utils.**send**(*recipients: Dict[str, List[int]], body: str, message_sent*)

Send Sebby sms

Parameters

- **recipients** – Dict {'<app_label>.<model>': [<id>, <id>]}
- **body** – Body of message
- **message_sent** – MessageSent instance

Returns MessageSent instance

Tasks

Storage

Send Utils

django_webix_sender.send_methods.storage.send_utils.**send**(*recipients: Dict[str, List[int]], subject: str, body: str, message_sent*)

Send email

Parameters

- **recipients** – Dict {'<app_label>.<model>': [<id>, <id>]}
- **subject** – Subject of message
- **body** – Body of message
- **message_sent** – MessageSent instance

Returns MessageSent instance

Telegram

Handlers

django_webix_sender.send_methods.telegram.handlers.**cancel_phone_number**(*update: telegram.update.Update, context: telegram.ext.callbackcontext.Cal* → int

Cancel phone number registration procedure

django_webix_sender.send_methods.telegram.handlers.**check_phone_number**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.Callb* → int

Check if user's phone number is in recipients records

django_webix_sender.send_methods.telegram.handlers.**check_user**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext* → None

Check if user is enabled to chat with bot

django_webix_sender.send_methods.telegram.handlers.**echo**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

Echo the user message.

django_webix_sender.send_methods.telegram.handlers.**help_command**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackCont* → None

Send a message when the command /help is issued.

django_webix_sender.send_methods.telegram.handlers.**start**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

Send a message when the command /start is issued.

django_webix_sender.send_methods.telegram.handlers.**start_phone_number**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.Callb* → int

Initial command that requires phone number to register user

Persistence

class django_webix_sender.send_methods.telegram.persistences.**DatabaseTelegramPersistence**(*st*
bo
=
Tr
st
bo
=
Tr
st
bo
=
Tr

get_bot_data() → Dict[Any, Any]
"Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the bot_data if stored, or an empty `dict`.

Returns: `dict`: The restored bot data.

get_chat_data() → DefaultDict[int, Dict[Any, Any]]
"Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the chat_data if stored, or an empty `defaultdict(dict)`.

Returns: `defaultdict`: The restored chat data.

get_conversations(*name: str*) → Dict[Tuple[int, ...], Optional[object]]
"Will be called by `telegram.ext.Dispatcher` when a `telegram.ext.ConversationHandler` is added if `telegram.ext.ConversationHandler.persistent` is `True`. It should return the conversations for the handler with *name* or an empty `dict`

Args: name (`str`): The handlers name.

Returns: `dict`: The restored conversations for the handler.

get_user_data() → DefaultDict[int, Dict[Any, Any]]
Will be called by `telegram.ext.Dispatcher` upon creation with a persistence object. It should return the user_data if stored, or an empty `defaultdict(dict)`.

Returns: `defaultdict`: The restored user data.

update_bot_data(*data: Dict[KT, VT]*) → None
Will be called by the `telegram.ext.Dispatcher` after a handler has handled an update.

Args: data (`dict`): The `telegram.ext.dispatcher.bot_data`.

update_chat_data(*chat_id: int, data: Dict[KT, VT]*) → None
Will be called by the `telegram.ext.Dispatcher` after a handler has handled an update.

Args: chat_id (`int`): The chat the data might have been changed for. data (`dict`): The `telegram.ext.dispatcher.chat_data` [chat_id].

update_conversation(*name: str, key: Tuple[int, ...], new_state: Optional[object]*) → None
Will be called when a `telegram.ext.ConversationHandler.update_state` is called. This allows the storage of the new state in the persistence.

Args: name (`str`): The handler's name. key (`tuple`): The key the state is changed for. new_state (`tuple | any`): The new state for the given key.

update_user_data(*user_id: int, data: Dict[KT, VT]*) → None
Will be called by the `telegram.ext.Dispatcher` after a handler has handled an update.

Args: user_id (`int`): The user the data might have been changed for. data (`dict`): The `telegram.ext.dispatcher.user_data` [user_id].

Send Utils

`django_webix_sender.send_methods.telegram.send_utils.`**send**(*recipients: Dict[str, List[int]], body: str, message_sent*)

Send Telegram message

Parameters

- **recipients** – Dict {'<app_label>.<model>': [<id>, <id>]}
- **body** – Body of message
- **message_sent** – MessageSent instance

Returns MessageSent instance

1.6 Change Log

All notable changes to this project will be documented in this file.

The format is based on KeepAChangelog and this project adheres to SemanticVersioning.

1.6.1 [Unreleased]

Added

- Added Telegram support
- New Skebby gateway APIs
- New storage send method
- Messages sent list
- Messages sent chat
- Added *collapsed* recipients list configuration property
- Added *groups_can_send* configuration
- Added *extra* configuration

Changed

- Changed send_methods path
- Moved *CONFIG_SKEBBY* to Skebby *send_method* configuration
- Changed send window to allow multiple send methods at the same time

Fixed

- Cast skebby results as str from byte string and try to convert as dict
- Invoices list fixes

1.6.2 [1.0.0] - 2020-05-28

Added

- Added maintainability badge
- Added translations
- Added email attachment link

Changed

- *django-webix* min version v1.2.0
- Better invoice area
- Optimized count with exists

Removed

- Removed old skebby gateway

Fixed

- Fixed gateway utils without set sender as django app
- Fixed filters
- Fixed python3 compatibility
- Fixed multiselect split
- Fixed sender window typology autocomplete

1.6.3 [0.3.6] - 2019-04-19

- Added and/or in list filters

1.6.4 [0.3.5] - 2019-04-05

- Fixed parametric send method

1.6.5 [0.3.4] - 2019-03-29

- Fixed kwargs in *send_mixin*
- Added parametric sms functionality

1.6.6 [0.3.3] - 2019-03-28

- Split send function

1.6.7 [0.3.2] - 2019-03-28

- Fix migration 0003

1.6.8 [0.3.1] - 2019-02-27

- Fix Django 2.0 templatetags
- Added support to model *select_related*, *prefetch_related* and *filter* of sender querysets

1.6.9 [0.1.0] - 2018-08-XX

- First release on PyPI.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

django_webix_sender.send_methods.email.send_utils, 15
django_webix_sender.send_methods.skebby.enums, 16
django_webix_sender.send_methods.skebby.exceptions, 16
django_webix_sender.send_methods.skebby.gateway, 16
django_webix_sender.send_methods.skebby.send_utils, 20
django_webix_sender.send_methods.skebby.tasks, 20
django_webix_sender.send_methods.storage.send_utils, 20
django_webix_sender.send_methods.telegram.handlers, 20
django_webix_sender.send_methods.telegram.persistences, 22
django_webix_sender.send_methods.telegram.send_utils, 23
django_webix_sender.templatetags.field_type, 14
django_webix_sender.templatetags.send_methods_utils, 14
django_webix_sender.templatetags.verbose_name, 15

Index

A

add_contact() (*django_webix_sender.send_methods.skebby.gateway.Skebby.Contacts method*), 17

C

cancel_phone_number() (*in module django_webix_sender.send_methods.telegram.handlers*), 20
check_phone_number() (*in module django_webix_sender.send_methods.telegram.handlers*), 20
check_user() (*in module django_webix_sender.send_methods.telegram.handlers*), 21
Customer (*class in django_webix_sender.models*), 12
Customer.DoesNotExist, 12
Customer.MultipleObjectsReturned, 12
CustomerTypology (*class in django_webix_sender.models*), 13
CustomerTypology.DoesNotExist, 13
CustomerTypology.MultipleObjectsReturned, 13

D

dashboard() (*django_webix_sender.send_methods.skebby.gateway.Skebby.User method*), 19
DatabaseTelegramPersistence (*class in django_webix_sender.send_methods.telegram.persistences*), 22
django_webix_sender.send_methods.email.send_utils (*module*), 15
django_webix_sender.send_methods.skebby.enums (*module*), 16
django_webix_sender.send_methods.skebby.exceptions (*module*), 16
django_webix_sender.send_methods.skebby.gateway (*module*), 16
django_webix_sender.send_methods.skebby.send_utils (*module*), 20
django_webix_sender.send_methods.skebby.tasks (*module*), 20
django_webix_sender.send_methods.storage.send_utils (*module*), 20
django_webix_sender.send_methods.telegram.handlers (*module*), 20
django_webix_sender.send_methods.telegram.persistence (*module*), 22
django_webix_sender.send_methods.telegram.send_util (*module*), 23
django_webix_sender.templatetags.field_type (*module*), 14
django_webix_sender.templatetags.send_methods_utils (*module*), 14
django_webix_sender.templatetags.verbose_name (*module*), 15
DjangoWebixSender (*class in django_webix_sender.models*), 11

E

echo() (*in module django_webix_sender.send_methods.telegram.handlers*), 21
ExternalSubject (*class in django_webix_sender.models*), 13
ExternalSubject.DoesNotExist, 13
ExternalSubject.MultipleObjectsReturned, 13
ExternalSubjectTypology (*class in django_webix_sender.models*), 14
ExternalSubjectTypology.DoesNotExist, 14
ExternalSubjectTypology.MultipleObjectsReturned, 14

F

field_type() (*in module django_webix_sender.templatetags.field_type*), 14

G

get_bot_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
get_chat_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
get_conversations() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
get_email (*django_webix_sender.models.Customer attribute*), 12
get_email (*django_webix_sender.models.DjangoWebixSender attribute*), 11
get_email (*django_webix_sender.models.ExternalSubject attribute*), 13
get_email_fieldpath() (*django_webix_sender.models.Customer static method*), 12
get_email_fieldpath() (*django_webix_sender.models.DjangoWebixSender static method*), 11
get_email_fieldpath() (*django_webix_sender.models.ExternalSubject static method*), 13
get_email_related (*django_webix_sender.models.DjangoWebixSender attribute*), 11
get_filters_viewers() (*django_webix_sender.models.Customer class method*), 12
get_filters_viewers() (*django_webix_sender.models.DjangoWebixSender class method*), 11
get_filters_viewers() (*django_webix_sender.models.ExternalSubject class method*), 13
get_prefetch_related() (*django_webix_sender.models.DjangoWebixSender class method*), 11
get_representation() (*django_webix_sender.models.Customer class method*), 12
get_representation() (*django_webix_sender.models.DjangoWebixSender class method*), 11
get_representation() (*django_webix_sender.models.ExternalSubject class method*), 13
get_select_related() (*django_webix_sender.models.DjangoWebixSender class method*), 11
get_sent_rich_sms_statistics() (*django_webix_sender.send_methods.skebby.gateway.Skebby.SmsHistory method*), 17
get_sent_sms_history() (*django_webix_sender.send_methods.skebby.gateway.Skebby.SmsHistory method*), 17
get_sent_sms_to_recipient() (*django_webix_sender.send_methods.skebby.gateway.Skebby.SmsHistory method*), 17
get_sms (*django_webix_sender.models.Customer attribute*), 12
get_sms (*django_webix_sender.models.DjangoWebixSender attribute*), 11
get_sms (*django_webix_sender.models.ExternalSubject attribute*), 13
get_sms_fieldpath() (*django_webix_sender.models.Customer static method*), 12
get_sms_fieldpath() (*django_webix_sender.models.DjangoWebixSender static method*), 12
get_sms_fieldpath() (*django_webix_sender.models.ExternalSubject static method*), 13
get_sms_related (*django_webix_sender.models.DjangoWebixSender attribute*), 12
get_sms_state() (*django_webix_sender.send_methods.skebby.gateway.Skebby.SmsHistory method*), 17
get_telegram (*django_webix_sender.models.Customer attribute*), 13
get_telegram (*django_webix_sender.models.DjangoWebixSender attribute*), 12
get_telegram (*django_webix_sender.models.ExternalSubject attribute*), 13
get_telegram_fieldpath() (*django_webix_sender.models.Customer static method*), 13
get_telegram_fieldpath() (*django_webix_sender.models.DjangoWebixSender static method*), 12
get_telegram_fieldpath() (*django_webix_sender.models.ExternalSubject static method*), 14
get_telegram_related (*django_webix_sender.models.DjangoWebixSender attribute*), 12
get_user_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
get_verbose_field_name() (*in module django_webix_sender.templatetags.verbose_name*), 15

H

help_command() (*in module django_webix_sender.send_methods.telegram.handlers*), 21

I

is_chat_available() (*in module django_webix_sender.templatetags.send_methods_utils*), 14
is_list_available() (*in module django_webix_sender.templatetags.send_methods_utils*), 15

M

MessageAttachment (*class in django_webix_sender.models*), 14
MessageAttachment.DoesNotExist, 14
MessageAttachment.MultipleObjectsReturned, 14
MessageRecipient (*class in django_webix_sender.models*), 14
MessageRecipient.DoesNotExist, 14
MessageRecipient.MultipleObjectsReturned, 14
MessageSent (*class in django_webix_sender.models*), 14
MessageSent.DoesNotExist, 14
MessageSent.MultipleObjectsReturned, 14
MessageTypology (*class in django_webix_sender.models*), 14
MessageTypology.DoesNotExist, 14
MessageTypology.MultipleObjectsReturned, 14
MessageUserRead (*class in django_webix_sender.models*), 14
MessageUserRead.DoesNotExist, 14
MessageUserRead.MultipleObjectsReturned, 14
my_import() (*in module django_webix_sender.utils*), 15

R

reset_password() (*django_webix_sender.send_methods.skebby.gateway.Skebby.User method*), 19

S

save_attachments() (*in module django_webix_sender.models*), 11
send() (*in module django_webix_sender.send_methods.email.send_utils*), 15
send() (*in module django_webix_sender.send_methods.skebby.send_utils*), 20
send() (*in module django_webix_sender.send_methods.storage.send_utils*), 20
send() (*in module django_webix_sender.send_methods.telegram.send_utils*), 23
send_mixin() (*in module django_webix_sender.utils*), 15
send_parametric_sms() (*django_webix_sender.send_methods.skebby.gateway.Skebby.Sms method*), 18
send_sms() (*django_webix_sender.send_methods.skebby.gateway.Skebby method*), 18
session_key() (*django_webix_sender.send_methods.skebby.gateway.Sk method*), 16
Skebby (*class in django_webix_sender.send_methods.skebby.gateway*), 16
Skebby.Authentication (*class in django_webix_sender.send_methods.skebby.gateway*), 16
Skebby.Contacts (*class in django_webix_sender.send_methods.skebby.gateway*), 16
Skebby.ContactsGroups (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.LandingPages (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.ReceivedSMS (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.SmsBlacklist (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.SmsHistory (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.SmsSend (*class in django_webix_sender.send_methods.skebby.gateway*), 17
Skebby.Subaccount (*class in django_webix_sender.send_methods.skebby.gateway*), 19
Skebby.TPOA (*class in django_webix_sender.send_methods.skebby.gateway*), 19
Skebby.TwoFactorAuthentication (*class in django_webix_sender.send_methods.skebby.gateway*), 19
Skebby.User (*class in django_webix_sender.send_methods.skebby.gateway*), 19
SkebbyBoolean (*class in django_webix_sender.send_methods.skebby.enums*), 16
SkebbyEncoding (*class in django_webix_sender.send_methods.skebby.enums*), 16
SkebbyException, 16
SkebbyMessageType (*class in django_webix_sender.send_methods.skebby.enums*),

16
start() (*in module django_webix_sender.send_methods.telegram.handlers*), 21
start_phone_number() (*in module django_webix_sender.send_methods.telegram.handlers*), 21

T

TelegramPersistence (*class in django_webix_sender.models*), 14
TelegramPersistence.DoesNotExist, 14
TelegramPersistence.MultipleObjectsReturned, 14

U

update_bot_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
update_chat_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
update_conversation() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
update_user_data() (*django_webix_sender.send_methods.telegram.persistences.DatabaseTelegramPersistence method*), 22
user_can_send() (*in module django_webix_sender.templatetags.send_methods_utils*), 15
user_status() (*django_webix_sender.send_methods.skebby.gateway.Skebby.User method*), 19
user_token() (*django_webix_sender.send_methods.skebby.gateway.Skebby.Authentication method*), 16

V

verify_session() (*django_webix_sender.send_methods.skebby.gateway.Skebby.User method*), 19