
postcodepy_proxy Documentation

Release 1.0.0

F. Brekeveld

February 26, 2016

1	Installation	3
1.1	Introduction	3
1.2	Application	3
1.3	Download & Install	3
2	postcodepy proxy API	5
2.1	Django App to the postcode.nl REST-service	5
3	Example	9
3.1	Setup the Django environment	9
3.2	Add views	9
3.3	Add a HTML template	10
3.4	Add request routes	10
3.5	Alter project settings	11
4	Indices and tables	13

Contents:

Installation

1.1 Introduction

The `postcodepy_proxy` package offers a simple API for Django to the Postcode.nl API REST service. To use the Django `postcodepy_proxy` you will need an *access_key* and *access_secret*. For details check api.postcode.nl.

Note: This package is ONLY useful to be used with the postcode REST-service of api.postcode.nl.

1.2 Application

The simple combination of `postcode/housenumber` allows the retrieval of a rich set of address information. You can use this for instance in websites for orderhandling or registration purposes. Using AJAX a registration form can be autocompleted for several parts. For details take a look at the *Example*.

1.3 Download & Install

1.3.1 From pypi

Install the package with pip:

```
$ pip install django-postcodepy-proxy
```

1.3.2 From Github

```
$ git clone https://github.com/hootnot/django-postcodepy-proxy.git
$ cd django-postcodepy-proxy
$ python setup.py install
```

postcodepy proxy API

2.1 Django App to the postcode.nl REST-service

This app make use of the `postcodepy` module. For details, see: [postcode-api-wrapper.readthedocs.org](https://readthedocs.org/projects/postcode-api-wrapper/).

class `postcodepy_proxy.views.PostcodepyProxyView` (***kwargs*)

Bases: `django.views.generic.base.View`

PostcodeProxyView - base View class to render a postcode API response.

derive your own View from this class to render postcode API responses.

__init__ (***kwargs*)

Constructor. Called in the URLconf; can contain helpful extra keyword arguments, and other things.

as_view (***initkwargs*)

Main entry point for a request-response process.

get (*request, postcode=None, houseNumber=None, houseNumberAddition=None*)

get - fetch 'adres-info' by 'postcode/huisnummer'.

Parameters

- **request** (*request*) – the django request object
- **postcode** (*str*) – postcode formatted as 4 digits 2 characters (no space), ex: 7514BP
- **houseNumber** (*int*) – housenumber without additions
- **houseNumberAddition** (*str (optional)*) – the housenumber addition as a string. Mostly one character (7514BP,129,A), but also numerical additions may apply (8651AP,1,41).

options (*request, *args, **kwargs*)

Handles responding to requests for the OPTIONS HTTP verb.

class `postcodepy_proxy.views.SignalProxyView` (***kwargs*)

Bases: `django.views.generic.base.View`

SignalProxyView - base View class to render a signal API response.

derive your own View from this class to render signal API responses.

__init__ (***kwargs*)

Constructor. Called in the URLconf; can contain helpful extra keyword arguments, and other things.

as_view (***initkwargs*)

Main entry point for a request-response process.

get (*request*, *sar*)

get - fetch 'signal-info' based on specified set of parameters.

Parameters *sar* (*dict*) – the signal-api-request (*sar*), being a dictionary of parameters relevant for the request. Since all parameters are optional, please take a look at the API documentation: <https://api.postcode.nl/documentation>

options (*request*, **args*, ***kwargs*)

Handles responding to requests for the OPTIONS HTTP verb.

class `postcodepy_proxy.signalapi.SignalRequestData` (*_d=None*, *delimiter=None*)

Bases: `object`

`SignalRequestData` - map flat formatted structure to nested structure.

Class to map for instance posted form data to a nested dictionary structure. This can be done when fields are named using a convention like: `name1_name2_name3`, but also: `name1.name2.name3`.

Example:

```
sarArgs = {
    "customer_email": "test-address@postcode.nl",
    "customer_phoneNumber": "+31235325689",
    "customer_address_postcode": "2012ES",
    "customer_address_houseNumber": "30",
    "customer_address_country": "NL",
    "transaction_internalId": "MyID-249084",
    "transaction_deliveryAddress_postcode": "7514BP",
    "transaction_deliveryAddress_houseNumber": "129",
    "transaction_deliveryAddress_houseNumberAddition": "B",
    "transaction_deliveryAddress_country": "NL"
}
```

with:

```
print (json.dumps(SignalRequestData(sarArgs)(),
    sort_keys=True, indent=2))
```

will be mapped to:

```
{
    "customer": {
        "address": {
            "country": "NL",
            "houseNumber": "30",
            "postcode": "2012ES"
        },
        "email": "test-address@postcode.nl",
        "phoneNumber": "+31235325689"
    },
    "transaction": {
        "deliveryAddress": {
            "country": "NL",
            "houseNumber": "129",
            "houseNumberAddition": "B",
            "postcode": "7514BP"
        },
        "internalId": "MyID-249084"
    }
}
```

`__init__(_d=None, delimiter=None)`

Instantiate a SignalRequestData object.

Parameters `d` ((dict) optional) –

Example

Lets create a simple Django application that accepts parameters via the URL and present the postcode.nl REST API response in HTML output.

3.1 Setup the Django environment

Create a virtual environment and install the packages we need:

```
$ cd <somedir>
$ virtualenv pcp
$ cd pcp
$ . ./bin/activate
$ pip install django
$ pip install django-postcodepy-proxy

$ django-admin startproject pcp
$ cd pcp
$ django-admin startapp pcproxy
$ cd pcproxy
```

3.2 Add views

Edit the file: pcproxy/views.py and add the HTML and JSON views.

```
from django.shortcuts import render
from django.http import HttpResponse
import json

from postcodepy_proxy.views import PostcodepyProxyView
from postcodepy.postcodepy import PostcodeError

class PCDemoHTMLView(PostcodepyProxyView):
    template_name = "postcodeproxy.html"

    def get(self, request, *args, **kwargs):
        rv = None
        try:
            rv = super(PCDemoHTMLView, self).get(request, *args, **kwargs)
```

```

    except PostcodeError as e:
        # Pass the exception information as response data
        rv = e.response_data

    return render(request, self.template_name, rv)

class PCDemoJSONView(PostcodepyProxyView):
    def get(self, request, *args, **kwargs):
        rv = None
        try:
            rv = super(PCDemoJSONView, self).get(request, *args, **kwargs)
        except PostcodeError as e:
            # Pass the exception information as response data
            rv = e.response_data

    return HttpResponse(json.dumps(rv, sort_keys=True, indent=2),
                        content_type="application/json")

```

3.3 Add a HTML template

Create a file: templates/postcodeproxy.html

```

<h1>Postcodeproxy</h1>
{{ postcode }}
{{ huisnummer }}
{{ city }}
{{ street }}
{{ postcode }}
{{ houseNumber }}
{{ latitude }}
{{ longitude }}

{% if exception %}
<h1>OOPS:<h1>
<h3>exception: {{exception}}<h3>
<h3>exceptionId: {{exceptionId}}<h3>
{% endif %}

```

3.4 Add request routes

Create the pcproxy/urls.py file and add url's to route the requests.

```

from django.conf.urls import url
from django.contrib import admin

from pcproxy import views

urlpatterns = [
    url(r'^postcode/ (?P<postcode>[\d]{4}[a-zA-Z]{2}) /'
        '(?P<houseNumber>[\d]+)/$',
        views.PCDemoHTMLView.as_view() ),
    url(r'^postcode/ (?P<postcode>[\d]{4}[a-zA-Z]{2}) /'

```

```

    '(?P<houseNumber>[\d]+)/'
    '(?P<houseNumberAddition>[\dA-Za-z]+)/$',
    views.PCDemoHTMLView.as_view() ),

    url(r'^jsonpostcode/(?P<postcode>[\d]{4}[a-zA-Z]{2})/'
        '(?P<houseNumber>[\d]+)/$',
        views.PCDemoJSONView.as_view() ),
    url(r'^jsonpostcode/(?P<postcode>[\d]{4}[a-zA-Z]{2})/'
        '(?P<houseNumber>[\d]+)/'
        '(?P<houseNumberAddition>[\dA-Za-z]+)/$',
        views.PCDemoJSONView.as_view() ),
]

```

3.5 Alter project settings

Edit the `pcp/settings.py` and add the apps

```

INSTALLED_APPS = (
    ...
    'postcodepy_proxy',
    'pcproxy',
)

```

and the authentication information required by *postcodepy_proxy*

```

POSTCODEPY = {
    "AUTH" : {
        "API_ACCESS_KEY" : "<your_access_key>",
        "API_ACCESS_SECRET" : "<your_access_secret>",
    },
}

```

3.5.1 Add the app urls to the project urls

Edit the project `pcp/urls.py` file and add the reference the `pcproxy/urls.py` file:

```

urlpatterns = [
    ...
    url(r'^pcp/', include('pcproxy.urls')),
]

```

3.5.2 Up and running ...

```
$ python manage.py runserver
```

From your webbrowser hit: `http://127.0.0.1:8000/pcp/jsonpostcode/7514BP/129/` and you should get the response:

```

{
  "addressType": "building",
  "bagAddressableObjectid": "0153010000345343",
  "bagNumberDesignationId": "0153200000345342",
  "city": "Enschede",

```

```
"houseNumber": 129,
"houseNumberAddition": "",
"houseNumberAdditions": [
    "",
    "A"
],
"latitude": 52.22770127,
"longitude": 6.89701549,
"municipality": "Enschede",
"postcode": "7514BP",
"province": "Overijssel",
"purposes": [
    "assembly"
],
"rdX": 258149,
"rdY": 472143,
"street": "Iasondersingel",
"surfaceArea": 6700
}
```

As you can see in the response, this postcode/number combination also comes with a houseNumberAddition A. When we hit: `http://127.0.0.1:8000/pcp/jsonpostcode/7514BP/129/A/` you should get the response:

```
{
  "addressType": "building",
  "bagAddressableObjectId": "0153010000329929",
  "bagNumberDesignationId": "0153200000329928",
  "city": "Enschede",
  "houseNumber": 129,
  "houseNumberAddition": "A",
  "houseNumberAdditions": [
    "",
    "A"
  ],
  "latitude": 52.22770127,
  "longitude": 6.89701549,
  "municipality": "Enschede",
  "postcode": "7514BP",
  "province": "Overijssel",
  "purposes": [
    "residency"
  ],
  "rdX": 258149,
  "rdY": 472143,
  "street": "Iasondersingel",
  "surfaceArea": 119
}
```

... or with an exception, hit: `http://127.0.0.1:8000/pcp/jsonpostcode/7514BP/129/B` and you should get the response:

```
{
  "exception": "Invalid housenumber addition: 'None'",
  "exceptionId": "ERRHouseNumberAdditionInvalid",
  "validHouseNumberAdditions": [
    "",
    "A"
  ]
}
```

Indices and tables

- `genindex`
- `modindex`
- `search`

Symbols

`__init__()` (postcodepy_proxy.signalapi.SignalRequestData
method), 6
`__init__()` (postcodepy_proxy.views.PostcodepyProxyView
method), 5
`__init__()` (postcodepy_proxy.views.SignalProxyView
method), 5

A

`as_view()` (postcodepy_proxy.views.PostcodepyProxyView
method), 5
`as_view()` (postcodepy_proxy.views.SignalProxyView
method), 5

G

`get()` (postcodepy_proxy.views.PostcodepyProxyView
method), 5
`get()` (postcodepy_proxy.views.SignalProxyView
method), 5

O

`options()` (postcodepy_proxy.views.PostcodepyProxyView
method), 5
`options()` (postcodepy_proxy.views.SignalProxyView
method), 6

P

`PostcodepyProxyView` (class in post-
codepy_proxy.views), 5

S

`SignalProxyView` (class in postcodepy_proxy.views), 5
`SignalRequestData` (class in post-
codepy_proxy.signalapi), 6