Django - OnMyDesk Documentation

Release 0.1.1

Alisson R. Perez (at Knowledge4Life)

September 22, 2016

Contents

1 Table of contents **3**

1.1 User guide . . . 3
1.1.1 Installation . . . 3
1.1.2 Quickstart . . . 3
1.1.3 A little bit more . . . 4
1.2 Schedulers . . . 5
1.2.1 Basic Usage . . . 5
1.2.2 Schedulers for reports with parameters . . . 6
1.3 Commands . . . 6
1.3.1 process . . . 6
1.3.2 scheduler_process . . . 7
1.4 Settings . . . 7
1.4.1 ONMYDESK_REPORT_LIST . . . 7
1.4.2 ONMYDESK_FILE_HANDLER . . . 7
1.4.3 ONMYDESK_DOWNLOAD_LINK_HANDLER . . . 8
1.4.4 ONMYDESK_NOTIFY_FROM . . . 8
1.4.5 ONMYDESK_SCHEDULER_NOTIFY_SUBJECT . . . 8
1.5 Reference . . . 8
1.5.1 onmydesk.models . . . 8
1.5.2 onmydesk.utils . . . 10
1.5.3 onmydesk.core.datasets . . . 10
1.5.4 onmydesk.core.outputs . . . 12
1.5.5 onmydesk.core.reports . . . 13

2 Indices and tables **17**

Python Module Index **19**

A Django app to build reports in a simple way.

App focused on developers with an easy way to retrieve information from application or any other source making it available to be shared.

Main features:

- Easy way to create reports by developers.
- Easy way to make reports available to users by a screen on admin.
- Possibility to use raw SQL queries or Django ORM to extract information.
- Possibility of retrieving informations from third party systems that enable some kind of integration (like an API).
- Easy way to give parameters from users on report creation.
- Possibility to schedule report creation with e-mail notification.

CHAPTER 1

Table of contents

1.1 User guide

1.1.1 Installation

With pip:

```
pip install django-onmydesk
```

Add 'onmydesk' to your INSTALLED_APPS:

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

Run *./manage.py migrate* to create **OnMyDesk** models.

1.1.2 Quickstart

To create reports we need to follow just two steps:

1. Create a report class in our django app.
2. Add this report class to a config in you project settings to enable **OnMyDesk** to see your reports.

So, let's do it!

Create a module called *reports.py* in you django app with the following content:

myapp/reports.py:

```
from onmydesk.core import reports

class UsersReport(reports.SQLReport):
    name = 'Users report'
    query = 'SELECT * FROM auth_user'
```

On your project settings, add the following config:

```
ONMYDESK_REPORT_LIST = [
    'myapp.reports.UsersReport',
]
```

Note: Each new report must be added to this list. Otherwise, it won't be shown on admin screen.

Now, access your **OnMyDesk** admin screen and you'll see your **Users report** available on report creation screen.

After you create a report, it'll be status settled up as 'Pending', to process it you must run *process* command. E.g:

```
$ ./manage.py process
Found 1 reports to process
Processing report #29 - 1 of 1
Report #29 processed
```

1.1.3 A little bit more

Parameters from user

Sometimes we need to get some info from user, like a date range, a category or something like this.

To do this, we can create a Django form to our report that will be showed to users in report selection on admin screen. After this, users will fill the form and the validated params will be available at report params attribute.

For example, with this changes our *report.py* should be like this:

```
from onmydesk.core import reports

from django import forms
from django.contrib.admin.widgets import AdminDateWidget


class UsersForm(forms.Form):
    start_date = forms.DateField(widget=AdminDateWidget)
    end_date = forms.DateField(widget=AdminDateWidget)


class UsersReport(reports.SQLReport):
    form = UsersForm

    name = 'Users report'

    query = 'SELECT * FROM auth_user WHERE date_joined between %s and %s'

    # Optionally we can use a db alias to change our database.
    db_alias = 'my-other-database'

    @property
    def query_params(self):
        return [
            self.params['start_date'],
            self.params['end_date']
        ]
```

PS.: We have a property called *query_params* in SQLReport that must return the params to be used in our query.

Other ways to get data

We aren't restricted by raw database sql queries (and we know that, by some reasons, it is not a good way to get our report data).

Reports in **OnMyDesk** are composed by Datasets and Outputs (we'll take a better look on both ahead). So, if you need to get your data by your own way you can create a Dataset. E.g.:

```
from onmydesk.core import reports, datasets, outputs


class TotalsDataset(datasets.BaseDataset):

    def iterate(self, params=None):
        return [
            ('Users', self._get_total_users()),
            ('Premium users', self._get_total_premium_users()),
        ]

    def _get_total_users(self):
        return 42  # Get your number from some source...

    def _get_total_premium_users(self):
        return 32  # Get your number from some source...


class TotalsReport(reports.BaseReport):
    name = 'Users - Totals'

    # Our report must be a csv file
    outputs = (outputs.CSVOutput(),)

    # An instance from our dataset
    dataset = TotalsDataset()
```

We just need to return an interable object in iterate method from our dataset.

Changing report's outputs

We can change the output of our report. The easier way to do this is to change *outputs* attribute of our report class.

Example:

```
class TotalsReport(reports.BaseReport):
    name = 'Users - Totals'

    # Changing our outputs to TSV and XLSx
    outputs = (outputs.TSVOutput(), outputs.XLSXOutput())
```

We have some output options by default. See more about on *onmydesk.core.outputs*.

1.2 Schedulers

With a scheduler we can schedule (Wow!) a generation of a report with some perioditicy automatically, like every monday or from monday to friday all the week.

1.2.1 Basic Usage

First of all, we'll need a report to schedule. If you don't have one, create it and go back here (see more on User guide).

Now, to use schedulers you basically need to create a schedule entry with admin screen (see more about on *Creating schedulers*) and run command *scheduler_process*.

If you want to notify someone about reports generated with schedulers it's better to setup *ONMYDESK_NOTIFY_FROM* and *ONMYDESK_SCHEDULER_NOTIFY_SUBJECT*.

Creating schedulers

1. Go to **Schedulers** screen in your admin screen and try to add a new schedule. You can notice this screen is very similar to report screen creation with some extra fields.
2. Select the **Report** that you want to schedule.
3. Select the **Periodicity**. You'll see some options like **Every monday** or **Monday to Friday**. This field is used to determinate which days of week your report will be generated automatically.
4. Fill **Notify e-mails** with the e-mails separated by ',' if you want to notify someone (including you).

Processing schedulers

To process them we only must to run *scheduler_process*. It's better to setup a cron or something similar to run this command once in a day to process all schedulers that have a periodicity matching with that week day.

1.2.2 Schedulers for reports with parameters

We can use reports with parameters (see about on *Parameters from user*) with our schedulers. Similar when we select a report on report creation screen, here when you select a report the page is reloaded and you can fill out the parameters to process your report.

Scheduling reports with date fields in parameters

When you select a report on a scheduler with a date field in its parameters you'll must fill this fields in a different way. Instead of selecting the exactally date you need to use **D** as a current date of report generation and subtract or add many days you need. Ok, it sounds confusing, let's go to an example:

Suposing you have a report that needs to be created every monday and builds its information with data of last week. This report should have two parameters to store this dates. When you select this report, you'll fill **start_date** with "D-7" and **end_date** with "D-1".

With this setup, running our scheduler on **May 9 2016** (this is our **D**), parameters will be with **start_date** as **May 2 2016** (**D-7**, Monday before) and **end_date** as **May 8 2106** (**D-1**, last Sunday).

1.3 Commands

1.3.1 process

Command used to process reports. E.g.:

```
$ ./manage.py process
```

For each time you run this command it'll try to find a pending report to process. It's better to put it in a cron or something like that to run each minute.

1.3.2 scheduler_process

Command used to process schedulers (see more on Schedulers). E.g.:

```
$ ./manage.py scheduler_process
```

For each time you call this command it'll try to find a scheduler entry for that weekday and process it. So, it's better to run it just one time in a day.

1.4 Settings

1.4.1 ONMYDESK_REPORT_LIST

It must contains a list of reports to be available at admin screen.

Example:

```
ONMYDESK_REPORT_LIST = [
    # ...
    'myapp.reports.MyReport',
]
```

1.4.2 ONMYDESK_FILE_HANDLER

It's an optional setting. It must be used to indicate a function to be called to handle a file after its generation. This function will receive the report filepath and must return a filepath, a url or something like this. It's useful to move reports to another directory or to a cloud storage.

Example:

We create a function at any place to upload our report to an Amazon S3 bucket:

```
# myapp/utils.py

def report_s3_upload(filepath):
    bucket = get_bucket(settings.BUCKETS['reports'])

    now = timezone.now()

    key_name = '{}/{}/{}'.format(
        now.strftime('%Y'),
        now.strftime('%m-%d'),
        path.basename(filepath))

    key = bucket.new_key(key_name)
    key.set_contents_from_filename(filepath)

    return key.name
```

On our settings, we setup with:

```
ONMYDESK_FILE_HANDLER = 'myapp.utils.report_s3_upload'
```

Now, our reports will be uploaded to our bucket at Amazon S3 after its processing.

1.4.3 ONMYDESK_DOWNLOAD_LINK_HANDLER

It's an optional setting. It must be used to indicate a function to generate a link to download our report file. This function will receive the report filepath or what was returned by *ONMYDESK_FILE_HANDLER* and must return a url to download the report file.

Example:

In the same way showed by *ONMYDESK_FILE_HANDLER*, now our function will return a url to download our report from Amazon S3 bucket:

```
# myapp/utils.py

def get_report_s3_link(filepath):
  bucket = get_bucket(settings.BUCKETS['reports'])

  key = bucket.get_key(filepath)

  return key.generate_url(settings.REPORT_S3_LINK_LIFETIME)
```

On our settings, we setup with:

```
ONMYDESK_DOWNLOAD_LINK_HANDLER = 'myapp.utils.get_report_s3_link'
```

1.4.4 ONMYDESK_NOTIFY_FROM

Used to fill out **from** field of sent e-mails. Default is 'no-reply@nobody.com'. E.g.:

```
ONMYDESK_NOTIFY_FROM = 'no-reply@mycompany.com'
```

1.4.5 ONMYDESK_SCHEDULER_NOTIFY_SUBJECT

Used by scheduler (see more on Schedulers) as e-mail subject. You can use *{report_name}* on your string to use report name. Default is *OnMyDesk - Report - {report_name}*. E.g.:

```
ONMYDESK_SCHEDULER_NOTIFY_SUBJECT = 'My company - Scheduled report {report_name}'
```

1.5 Reference

1.5.1 onmydesk.models

Required models to handle and store generated reports.

class onmydesk.models.**Report**(**args*, ***kwargs*)

Report model to store generated reports.

get_params()

Return param to be used to process report.

Returns Report params

process()

Process this report.

After processing the outputs will be stored at *results*. To access output results is recommended to use *results_as_list()*.

result_links
Return a list with links to access report results.

Returns List of links to access results

Return type list

results_as_list
Return a list of output results stored in this model.

Returns List of results

Return type list

set_params(*params*)
Set params to be used when report is processed.

Parameters **params** (*dict*) – Dictionary with params to be used to process report.

exception onmydesk.models.**ReportNotSavedException**
Exception used when a report is not saved.

class onmydesk.models.**Scheduler**(**args*, ***kwargs*)
Model used to schedule reports.

It'd be used to schedule reports to be generated with some periodicity (every monday, from monday to friday, etc.)

get_params()
Return params to be used to process report.

Returns Report params

get_processed_params(*reference_date=None*)
Return params to be used to process report.

Parameters **reference_date** (*date*) – Date to use as reference

Returns Dict with params

Return type dict

process(*reference_date=None*)
Process scheduler creating and returing a report.

After processing, this method tries to notify e-mails filled in notify_emails field.

Returns Report result

Return type *Report*

set_params(*params*)
Set params to be used when report is processed.

Parameters **params** (*dict*) – Dictionary with params to be used to process report.

onmydesk.models.**output_file_handler**(*filepath*)
Return the output filepath (handled or not by an external function).

This function tries to find a function handler in *settings.ONMYDESK_FILE_HANDLER*. It must receive a filepath and returns a new filepath (or url, e.g.) to be stored in the report register. It's useful to handle the report results (move to other dirs ou to cloud).

Parameters **filepath** (*str*) – File path to output generated by report.

Returns File path to output (processed or not by a external handler)

Return type str

1.5.2 onmydesk.utils

Module with common utilities to this package.

onmydesk.utils.**my_import**(*class_name*)
Return a python class given a class name.

Usage example:

```
Report = my_import('myclass.models.Report')

model_instance = Report()
model_instance.name = 'Test'
model_instance.save()
```

Parameters **class_name** (*str*) – Class name

Returns Class object

onmydesk.utils.**str_to_date**(*value*, *reference_date*)
Convert a string like 'D-1' to a "reference_date - timedelta(days=1)".

Parameters

- **value** (*str*) – String like 'D-1', 'D+1', 'D'...
- **reference_date** (*date*) – Date to be used as 'D'

Returns Result date

Return type date

onmydesk.utils.**with_metaclass**(*mcls*)
Decorator used to keep compatibility between python 2 and 3.

E.g.:

```
@with_metaclass(ABCMeta)
class MyClass(object):
    pass
```

1.5.3 onmydesk.core.datasets

Datasets from library.

Datasets are used to get data from any source and return them to reports in a simple way.

class onmydesk.core.datasets.**BaseDataset**
An abstract representation of what must be a Dataset class.

It's possible to use context management with datasets. To do this you must override methods *__enter__()* to lock resources and *__exit__()* to free up them. E.g.:

```
class MyDataset(BaseDataset):
    def iterate(self, params=None):
        return self.file.read()

    def __enter__(self):
```

```
            self.file = open('somefile.txt')

        def __exit__(self, type, value, traceback):
            self.file.close()

    with MyDataset() as mydataset:
        for row in mydataset.iterate():
            print(row)
```

__enter__()
Enter from context manager to lock some resource (for example).

__exit__(*type*, *value*, *traceback*)
Exit from context manager to free up some resource (for example).

__weakref__
list of weak references to the object (if defined)

iterate(*params=None*)
Return an iterable object.

Parameters **params** (*dict*) – Parameters to be used by dataset

class onmydesk.core.datasets.**SQLDataset**(*query*, *query_params=[]*, *db_alias=None*)
A SQLDataset is used to run raw queries into database.

E.g.:

```
    with SQLDataset('SELECT * FROM users'):
        for row in mydataset.iterate():
            print(row)    # --> A OrderedDict with cols and values.
```

Note: It's recomended to use instances of this class using *with* statement.

BE CAREFUL

Always use *query_params* from *__init__()* to put dinamic values into the query. E.g.:

```
    # WRONG WAY:
    mydataset = SQLDataset('SELECT * FROM users where age > {}'.format(18))

    # RIGHT WAY:
    mydataset = SQLDataset('SELECT * FROM users where age > %d', [18])
```

__enter__()
Enter from context manager to open a cursor with database.

__exit__(*type*, *value*, *traceback*)
Exit from context manager to close cursor with database.

__init__(*query*, *query_params=[]*, *db_alias=None*)
Init method.

Parameters

- **query** (*str*) – Raw sql query.
- **query_params** (*list*) – Params to be evaluated with query.
- **db_alias** (*str*) – Database alias from django settings. Optional.

iterate(*params=None*)
Return an iterable rows (ordered dicts).

Parameters **params** (*dict*) – Parameters to be used by dataset.

Returns Rows from query result.

Return type Iterator with OrderedDict items.

1.5.4 onmydesk.core.outputs

Outputs from library.

Outputs are used to get data and convert it to some representation (like a CSV or XLSX).

class onmydesk.core.outputs.**BaseOutput**
An abstract representation of an Output class.

We need to use instances of this classes with context manager. E.g.:

```
from onmydesk.core.outputs import XLSXOutput

myout = XLSXOutput()

with myout as output:
    output.header(['Name', 'Age'])
    output.out(['Alisson', 39])
    output.out(['Joao', 16])
    output.footer(['Total', 55])
    print(output.filepath)
```

__enter__()
Used by a context manager.

__exit__(*exc_type*, *exc_value*, *traceback*)
Used by context manager to exit object.

__init__()
Class initializer.

__weakref__
list of weak references to the object (if defined)

file_extension = None
File extension to be used on file name output. E.g.: 'csv'

filepath = None
Filepath with output result which is filled by process().

footer(*content*)
Output a footer content.

Parameters **content** (*mixed*) – Content to be written

gen_tmpfilename()
Utility to be used to generate a temporary filename.

Returns Temporary filepath.

Return type str

header(*content*)
Return output a header content.

Parameters **content** (*mixed*) – Content to be written

name = None
Name used to compose output filename

out(*content*)
Output a normal content.

Parameters **content** (*mixed*) – Content to be written

class onmydesk.core.outputs.**CSVOutput**(**args*, ***kwargs*)
An output to generate CSV files (files with cols separated by comma).

class onmydesk.core.outputs.**SVOutput**(**args*, ***kwargs*)
Abstract separated values output.

__enter__()
Enter from context manager.

__exit__(**args*, ***kwargs*)
Exit from context manager.

__init__(**args*, ***kwargs*)
Class initializer.

out(*content*)
Output a content to a separated value line.

Parameters **content** (*mixed*) – Content to be written

class onmydesk.core.outputs.**TSVOutput**(**args*, ***kwargs*)
An output to generate TSV files (files with cols separated by tabs).

class onmydesk.core.outputs.**XLSXOutput**
Output to generate XLSX files.

__enter__()
Enter from a context manager.

__exit__(**args*, ***kwargs*)
Exit from context manager.

footer(*content*)
Output a footer content.

Parameters **content** (*mixed*) – Content to be written

header(*content*)
Output a header content.

Parameters **content** (*mixed*) – Content to be written

min_width = 8.43
Min width used to set column widths

out(*content*)
Output a normal content.

Parameters **content** (*mixed*) – Content to be written

1.5.5 onmydesk.core.reports

Reports from library.

Base entity from library, used to centralize outputs and datasets.

class onmydesk.core.reports.**BaseReport**(*params=None*)
An abstract representation of a report.

__init__(*params=None*)
Class initializer.

Parameters **params** (*dict*) – Params to be used by report (a date range to fetch data from database, for example).

__weakref__
list of weak references to the object (if defined)

_write_content(*outputs*, *items*)
Write a normal content in outputs.

Parameters

- **outputs** (*list*) – A list of output objects.
- **items** (*iterable*) – Itens (rows) to be written in outputs.

_write_footer(*outputs*)
Write a footer content in outputs.

Parameters **outputs** (*list*) – A list of output objects.

_write_header(*outputs*)
Write a header in outputs.

Parameters **outputs** (*list*) – A list of output objects.

dataset
Return Dataset to be used by this report.

Returns Dataset to be used by this report.

footer = None
Report footer.

form = None
Django form class to enable user to fill some param.

classmethod get_form()
Return form to be used with this report in admin creation screen.

Returns Form to be used with this report in admin creation screen.

header = None
Report header.

name = None
Report name. E.g.: *Monthly sales*.

output_filepaths = []
Output files filled by *process()*.

outputs
Return a list of outputs to be used by this report.

Returns A list of outputs to be used by this report.

params = None
Report params, it's used to process report.

process()
Process report and store output filepaths in *output_filepaths*.

row_cleaner(*row*)
Method used to handle line by line of the report.

It's useful to convert some data or do some sanitization.

Parameters **row** – Line to be rendered in the report.

Returns Line after some processing with it.

class onmydesk.core.reports.**SQLReport**(*params=None*)
Report to be used with raw SQL's.

E.g.:

```
class SalesReport(SQLReport):
    query = 'SELECT * FROM sales'

report = SalesReport()
report.process()

print(report.output_filepaths) # --> Files with all rows from sales table.
```

dataset
Return SQLDataset to be used by this report.

db_alias = None
Database alias from django config to be used with queries

outputs = (<onmydesk.core.outputs.TSVOutput object at 0x7fa13bf23ac8>,)
Outputs list, default TSV.

query = None
Raw report query.

query_params = []
Params to be evaluated with query.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

O

onmydesk.core.datasets, 10
onmydesk.core.outputs, 12
onmydesk.core.reports, 13
onmydesk.models, 8
onmydesk.utils, 10

Symbols

__enter__() (onmydesk.core.datasets.BaseDataset method), 11
__enter__() (onmydesk.core.datasets.SQLDataset method), 11
__enter__() (onmydesk.core.outputs.BaseOutput method), 12
__enter__() (onmydesk.core.outputs.SVOutput method), 13
__enter__() (onmydesk.core.outputs.XLSXOutput method), 13
__exit__() (onmydesk.core.datasets.BaseDataset method), 11
__exit__() (onmydesk.core.datasets.SQLDataset method), 11
__exit__() (onmydesk.core.outputs.BaseOutput method), 12
__exit__() (onmydesk.core.outputs.SVOutput method), 13
__exit__() (onmydesk.core.outputs.XLSXOutput method), 13
__init__() (onmydesk.core.datasets.SQLDataset method), 11
__init__() (onmydesk.core.outputs.BaseOutput method), 12
__init__() (onmydesk.core.outputs.SVOutput method), 13
__init__() (onmydesk.core.reports.BaseReport method), 14
__weakref__ (onmydesk.core.datasets.BaseDataset attribute), 11
__weakref__ (onmydesk.core.outputs.BaseOutput attribute), 12
__weakref__ (onmydesk.core.reports.BaseReport attribute), 14
_write_content() (onmydesk.core.reports.BaseReport method), 14
_write_footer() (onmydesk.core.reports.BaseReport method), 14
_write_header() (onmydesk.core.reports.BaseReport method), 14

B

BaseDataset (class in onmydesk.core.datasets), 10
BaseOutput (class in onmydesk.core.outputs), 12
BaseReport (class in onmydesk.core.reports), 14

C

CSVOutput (class in onmydesk.core.outputs), 13

D

dataset (onmydesk.core.reports.BaseReport attribute), 14
dataset (onmydesk.core.reports.SQLReport attribute), 15
db_alias (onmydesk.core.reports.SQLReport attribute), 15

F

file_extension (onmydesk.core.outputs.BaseOutput attribute), 12
filepath (onmydesk.core.outputs.BaseOutput attribute), 12
footer (onmydesk.core.reports.BaseReport attribute), 14
footer() (onmydesk.core.outputs.BaseOutput method), 12
footer() (onmydesk.core.outputs.XLSXOutput method), 13
form (onmydesk.core.reports.BaseReport attribute), 14

G

gen_tmpfilename() (onmydesk.core.outputs.BaseOutput method), 12
get_form() (onmydesk.core.reports.BaseReport class method), 14
get_params() (onmydesk.models.Report method), 8
get_params() (onmydesk.models.Scheduler method), 9
get_processed_params() (onmydesk.models.Scheduler method), 9

H

header (onmydesk.core.reports.BaseReport attribute), 14

header() (onmydesk.core.outputs.BaseOutput method), 12
header() (onmydesk.core.outputs.XLSXOutput method), 13

I

iterate() (onmydesk.core.datasets.BaseDataset method), 11
iterate() (onmydesk.core.datasets.SQLDataset method), 11

M

min_width (onmydesk.core.outputs.XLSXOutput attribute), 13
my_import() (in module onmydesk.utils), 10

N

name (onmydesk.core.outputs.BaseOutput attribute), 13
name (onmydesk.core.reports.BaseReport attribute), 14

O

onmydesk.core.datasets (module), 10
onmydesk.core.outputs (module), 12
onmydesk.core.reports (module), 13
onmydesk.models (module), 8
onmydesk.utils (module), 10
out() (onmydesk.core.outputs.BaseOutput method), 13
out() (onmydesk.core.outputs.SVOutput method), 13
out() (onmydesk.core.outputs.XLSXOutput method), 13
output_file_handler() (in module onmydesk.models), 9
output_filepaths (onmydesk.core.reports.BaseReport attribute), 14
outputs (onmydesk.core.reports.BaseReport attribute), 14
outputs (onmydesk.core.reports.SQLReport attribute), 15

P

params (onmydesk.core.reports.BaseReport attribute), 14
process() (onmydesk.core.reports.BaseReport method), 14
process() (onmydesk.models.Report method), 8
process() (onmydesk.models.Scheduler method), 9

Q

query (onmydesk.core.reports.SQLReport attribute), 15
query_params (onmydesk.core.reports.SQLReport attribute), 15

R

Report (class in onmydesk.models), 8
ReportNotSavedException, 9
result_links (onmydesk.models.Report attribute), 9
results_as_list (onmydesk.models.Report attribute), 9
row_cleaner() (onmydesk.core.reports.BaseReport method), 15

S

Scheduler (class in onmydesk.models), 9
set_params() (onmydesk.models.Report method), 9
set_params() (onmydesk.models.Scheduler method), 9
SQLDataset (class in onmydesk.core.datasets), 11
SQLReport (class in onmydesk.core.reports), 15
str_to_date() (in module onmydesk.utils), 10
SVOutput (class in onmydesk.core.outputs), 13

T

TSVOutput (class in onmydesk.core.outputs), 13

W

with_metaclass() (in module onmydesk.utils), 10

X

XLSXOutput (class in onmydesk.core.outputs), 13