# Django Data Importer Documentation

***Release 2.2.1***

**Valder Gallo**

August 19, 2015

Contents

Data importer documentation help

Contents:

CHAPTER 1

# Django Data Importer

**Django Data Importer** is a tool which allow you to transform easily a CSV, XML, XLS and XLSX file into a python object or a django model instance. It is based on the django-style declarative model.

## 1.1 Documentation and usage

Read docs online in Read the Docs:

https://django-data-importer.readthedocs.org/

You can generate everything at the above site in your local folder by:

```
$ cd doc
$ make html
$ open _build/html/index.html # Or your preferred web browser
```

## 1.2 Installation

Use either `easy_install`:

```
easy_install data-importer
```

or `pip`:

```
pip install data-importer
```

## 1.3 Settings

Customize data_importer decoders

**DATA_IMPORTER_EXCEL_DECODER** Default value is cp1252

**DATA_IMPORTER_DECODER** Default value is UTF-8

## 1.4 Basic example

Consider the following:

```
>>> from data_importer.importers import CSVImporter
>>> class MyCSVImporterModel(CSVImporter):
...     fields = ['name', 'age', 'length']
...     class Meta:
...         delimiter = ";"
```

You declare a `MyCSVImporterModel` which will match to a CSV file like this:

> Anthony;27;1.75

To import the file or any iterable object, just do:

```
>>> my_csv_list = MyCSVImporterModel(source="my_csv_file_name.csv")
>>> row, first_line = my_csv_list.cleaned_data[0]
>>> first_line['age']
27
```

Without an explicit declaration, data and columns are matched in the same order:

```
Anthony --> Column 0 --> Field 0 --> name
27      --> Column 1 --> Field 1 --> age
1.75    --> Column 2 --> Field 2 --> length
```

## 1.5 Django Model

If you now want to interact with a django model, you just have to add a **Meta.model** option to the class meta.

```
>>> from django.db import models
>>> class MyModel(models.Model):
...     name = models.CharField(max_length=150)
...     age = models.CharField(max_length=150)
...     length = models.CharField(max_length=150)
```

```
>>> from data_importer.importers import CSVImporter
>>> from data_importer.model import MyModel
>>> class MyCSVImporterModel(CSVImporter):
...     class Meta:
...         delimiter = ";"
...         model = MyModel
```

That will automatically match to the following django model.

*The django model should be imported in the model*

**class `Meta`**

> **delimiter** define the delimiter of the csv file. If you do not set one, the sniffer will try yo find one itself.
>
> **ignore_first_line** Skip the first line if True.
>
> **model** If defined, the importer will create an instance of this model.
>
> **raise_errors** If set to True, an error in a imported line will stop the loading.
>
> **exclude** Exclude fields from list fields to import

**transaction** ***(beta) not tested*** Use transaction to save objects

**ignore_empty_lines** Not validate empty lines

## 1.6 Django XML

If you now want to interact with a django model, you just have to add a **Meta.model** option to the class meta.

XML file example:

```
<encspot>
  <file>
   <Name>Rocky Balboa</Name>
   <Age>40</Age>
   <Height>1.77</Height>
  </file>
  <file>
   <Name>Chuck Norris</Name>
   <Age>73</Age>
   <Height>1.78</Height>
  </file>
</encspot>
```

```
>>> from django.db import models
>>> class MyModel(models.Model):
...     name = models.CharField(max_length=150)
...     age = models.CharField(max_length=150)
...     height = models.CharField(max_length=150)
```

```
>>> from data_importer.importers import XMLImporter
>>> from data_importer.model import MyModel
>>> class MyCSVImporterModel(XMLImporter):
...     root = 'file'
...     class Meta:
...         model = MyModel
```

That will automatically match to the following django model.

*The django model should be imported in the model*

**class Meta**

**model** If defined, the importer will create an instance of this model.

**raise_errors** If set to True, an error in a imported line will stop the loading.

**exclude** Exclude fields from list fields to import

**transaction** ***(beta) not tested*** Use transaction to save objects

## 1.7 Django XLS/XLSX

My XLS/XLSX file can be imported too

| Header1 | Header2 | Header3 | Header4 |
|---|---|---|---|
| Teste 1 | Teste 2 | Teste 3 | Teste 4 |
| Teste 1 | Teste 2 | Teste 3 | Teste 4 |

This is my model

```
>>> from django.db import models
>>> class MyModel(models.Model):
...     header1 = models.CharField(max_length=150)
...     header2 = models.CharField(max_length=150)
...     header3 = models.CharField(max_length=150)
...     header4 = models.CharField(max_length=150)
```

This is my class

```
>>> from data_importer.importers import XLSImporter
>>> from data_importer.model import MyModel
>>> class MyXLSImporterModel(XLSImporter):
...     class Meta:
...         model = MyModel
```

If you are using XLSX you will need use XLSXImporter to made same importer

```
>>> from data_importer.importers import XLSXImporter
>>> from data_importer.model import MyModel
>>> class MyXLSXImporterModel(XLSXImporter):
...     class Meta:
...         model = MyModel
```

**class `Meta`**

**ignore_first_line** Skip the first line if True.

**model** If defined, the importer will create an instance of this model.

**raise_errors** If set to True, an error in a imported line will stop the loading.

**exclude** Exclude fields from list fields to import

**transaction *(beta) not tested*** Use transaction to save objects

## 1.8 Descriptor

Using file descriptor to define fields for large models.

import_test.json

```
{
  'app_name': 'mytest.Contact',
    {
    // field name / name on import file or key index
    'name': 'My Name',
    'year': 'My Year',
    'last': 3
    }
}
```

model.py

```
class Contact(models.Model):
  name = models.CharField(max_length=50)
  year = models.CharField(max_length=10)
  laster = models.CharField(max_length=5)
  phone = models.CharField(max_length=5)
```

```
  address = models.CharField(max_length=5)
  state = models.CharField(max_length=5)
```

importer.py

```
class MyImpoter(BaseImpoter):
  class Meta:
    config_file = 'import_test.json'
    model = Contact
    delimiter = ','
    ignore_first_line = True
```

content_file.csv

```
name,year,last
Test,12,1
Test2,13,2
Test3,14,3
```

## 1.9 TEST

| | | |
|---|---|---|
| Acentuation with XLS | Excel MAC 2011 | OK |
| Acentuation with XLS | Excel WIN 2010 | OK |
| Acentuation with XLSX | Excel MAC 2011 | OK |
| Acentuation with XLSX | Excel WIN 2010 | OK |
| Acentuation with CSV | Excel Win 2010 | OK |

---

**Python** python 2.7

**Django** 1.6; 1.7

CHAPTER 2

# Core

## 2.1 Base

data_importer.core.base.**objclass2dict**(*objclass*)

Meta is a objclass on python 2.7 and no have __dict__ attribute.

This method convert one objclass to one lazy dict without AttributeError

## 2.2 Default Settings

Customize data_importer decoders

**DATA_IMPORTER_EXCEL_DECODER** Default value is cp1252

**DATA_IMPORTER_DECODER** Default value is UTF-8

**DATA_IMPORTER_TASK**

**Need Celery installed to set importers as Task** *default value is False*

**DATA_IMPORTER_QUEUE**

**Set Celery Queue in DataImpoter Tasks** *default value is DataImporter*

**DATA_IMPORTER_TASK_LOCK_EXPIRE**

**Set task expires time** *default value is 60 * 20*

## 2.3 Exceptions

**exception** data_importer.core.exceptions.**InvalidDescriptor**

Invalid Descriptor File Descriptor must be one valid JSON

**exception** data_importer.core.exceptions.**InvalidModel**

Invalid model in descriptor

**exception** data_importer.core.exceptions.**StopImporter**

Stop interator and raise error message

**exception** data_importer.core.exceptions.**UnsuportedFile**

Unsuported file type

## 2.4 Descriptors

**class** data_importer.core.descriptor.**ReadDescriptor**(*file_name=None*, *model_name=None*)

**get_fields**()
Get content

**get_model**()
Read model from JSON descriptor

**read_file**()
Read json file

CHAPTER 3

# Importers

## 3.1 XLSImporter

**class** data_importer.importers.xls_importer.**XLSImporter**(*source=None*, **args*, ***kwargs*)

**class Meta**
Importer configurations

XLSImporter.**clean**()
Custom clean method

XLSImporter.**clean_field**(*field_name*, *value*)
User default django field validators to clean content and run custom validates

XLSImporter.**clean_row**(*row_values*)
Custom clean method for full row data

XLSImporter.**cleaned_data**
Return tupla with data cleaned

XLSImporter.**errors**
Show errors catch by clean methods

XLSImporter.**exclude_fields**()
Exclude fields from Meta.exclude

XLSImporter.**get_error_message**(*error*, *row=None*, *error_type=None*)

XLSImporter.**is_valid**()
Clear content and return False if have errors

XLSImporter.**load_descriptor**()
Set fields from descriptor file

XLSImporter.**meta**
Is same to use .Meta

XLSImporter.**post_clean**()
Excuted after all clean method

XLSImporter.**post_save_all_lines**()
End exection

XLSImporter.**pre_clean**()
Executed before all clean methods Important: pre_clean dont have cleaned_data content

XLSImporter.**pre_commit**()
Executed before commit multiple register

XLSImporter.**process_row**(*row*, *values*)
Read clean functions from importer and return tupla with row number, field and value

XLSImporter.**save**(*instance=None*)
Save all contents DONT override this method

XLSImporter.**set_reader**()
[[1,2,3], [2,3,4]]

XLSImporter.**source**
Return source opened

XLSImporter.**start_fields**()
Initial function to find fields or headers values This values will be used to process clean and save method If this method not have fields and have Meta.model this method will use model fields to populate content without id

XLSImporter.**to_unicode**(*bytestr*)
Receive string bytestr and try to return a utf-8 string.

## 3.2 XLSImporter

**class** data_importer.importers.xlsx_importer.**XLSXImporter**(*source=None*, **args*, ***kwargs*)

**class Meta**
Importer configurations

XLSXImporter.**clean**()
Custom clean method

XLSXImporter.**clean_field**(*field_name*, *value*)
User default django field validators to clean content and run custom validates

XLSXImporter.**clean_row**(*row_values*)
Custom clean method for full row data

XLSXImporter.**cleaned_data**
Return tupla with data cleaned

XLSXImporter.**errors**
Show errors catch by clean methods

XLSXImporter.**exclude_fields**()
Exclude fields from Meta.exclude

XLSXImporter.**get_error_message**(*error*, *row=None*, *error_type=None*)

XLSXImporter.**is_valid**()
Clear content and return False if have errors

XLSXImporter.**load_descriptor**()
Set fields from descriptor file

XLSXImporter.**meta**
Is same to use .Meta

XLSXImporter.**post_clean**()
Excuted after all clean method

XLSXImporter.**post_save_all_lines**()
End exection

XLSXImporter.**pre_clean**()
Executed before all clean methods Important: pre_clean dont have cleaned_data content

XLSXImporter.**pre_commit**()
Executed before commit multiple register

XLSXImporter.**process_row**(*row*, *values*)
Read clean functions from importer and return tupla with row number, field and value

XLSXImporter.**save**(*instance=None*)
Save all contents DONT override this method

XLSXImporter.**set_reader**(*use_iterators=True*, *data_only=True*)
Read XLSX files

XLSXImporter.**source**
Return source opened

XLSXImporter.**start_fields**()
Initial function to find fields or headers values This values will be used to process clean and save method If this method not have fields and have Meta.model this method will use model fields to populate content without id

XLSXImporter.**to_unicode**(*bytestr*)
Receive string bytestr and try to return a utf-8 string.

## 3.3 XMLImporter

**class** data_importer.importers.xml_importer.**XMLImporter**(*source=None*, **args*, ***kwargs*)
Import XML files

**class Meta**
Importer configurations

XMLImporter.**clean**()
Custom clean method

XMLImporter.**clean_field**(*field_name*, *value*)
User default django field validators to clean content and run custom validates

XMLImporter.**clean_row**(*row_values*)
Custom clean method for full row data

XMLImporter.**cleaned_data**
Return tupla with data cleaned

XMLImporter.**errors**
Show errors catch by clean methods

XMLImporter.**exclude_fields**()
Exclude fields from Meta.exclude

XMLImporter.**get_error_message**(*error*, *row=None*, *error_type=None*)

XMLImporter.**is_valid**()
Clear content and return False if have errors

XMLImporter.**load_descriptor**()
Set fields from descriptor file

XMLImporter.**meta**
Is same to use .Meta

XMLImporter.**post_clean**()
Excuted after all clean method

XMLImporter.**post_save_all_lines**()
End exection

XMLImporter.**pre_clean**()
Executed before all clean methods Important: pre_clean dont have cleaned_data content

XMLImporter.**pre_commit**()
Executed before commit multiple register

XMLImporter.**process_row**(*row*, *values*)
Read clean functions from importer and return tupla with row number, field and value

XMLImporter.**root = 'root'**

XMLImporter.**save**(*instance=None*)
Save all contents DONT override this method

XMLImporter.**set_reader**()

XMLImporter.**source**
Return source opened

XMLImporter.**start_fields**()
Initial function to find fields or headers values This values will be used to process clean and save method If this method not have fields and have Meta.model this method will use model fields to populate content without id

XMLImporter.**to_unicode**(*bytestr*)
Receive string bytestr and try to return a utf-8 string.

## 3.4 GenericImporter

**class** data_importer.importers.generic.**GenericImporter**(*source=None*, **args*, ***kwargs*)
An implementation of BaseImporter that sets the right reader by file extension. Probably the best choice for almost all implementation cases

**class Meta**
Importer configurations

GenericImporter.**clean**()
Custom clean method

GenericImporter.**clean_field**(*field_name*, *value*)
User default django field validators to clean content and run custom validates

GenericImporter.**clean_row**(*row_values*)
Custom clean method for full row data

GenericImporter.**cleaned_data**
Return tupla with data cleaned

GenericImporter.**errors**
Show errors catch by clean methods

GenericImporter.**exclude_fields**()
Exclude fields from Meta.exclude

GenericImporter.**get_error_message**(*error*, *row=None*, *error_type=None*)

GenericImporter.**get_reader_class**()
Gets the right file reader class by source file extension

GenericImporter.**get_source_file_extension**()
Gets the source file extension. Used to choose the right reader

GenericImporter.**is_valid**()
Clear content and return False if have errors

GenericImporter.**load_descriptor**()
Set fields from descriptor file

GenericImporter.**meta**
Is same to use .Meta

GenericImporter.**post_clean**()
Excuted after all clean method

GenericImporter.**post_save_all_lines**()
End exection

GenericImporter.**pre_clean**()
Executed before all clean methods Important: pre_clean dont have cleaned_data content

GenericImporter.**pre_commit**()
Executed before commit multiple register

GenericImporter.**process_row**(*row*, *values*)
Read clean functions from importer and return tupla with row number, field and value

GenericImporter.**save**(*instance=None*)
Save all contents DONT override this method

GenericImporter.**set_reader**()

GenericImporter.**source**
Return source opened

GenericImporter.**start_fields**()
Initial function to find fields or headers values This values will be used to process clean and save method If this method not have fields and have Meta.model this method will use model fields to populate content without id

GenericImporter.**to_unicode**(*bytestr*)
Receive string bytestr and try to return a utf-8 string.

CHAPTER 4

# Data Importer Forms

Is one simple django.ModelForm with content to upload content

**Parameters**

- content (*FileField*) File uploaded

## 4.1 FileUploadForm

CHAPTER 5

# Data Importer Models

## 5.1 FileHistory

CHAPTER 6

# Views

## 6.1 DataImporterForm

Is a mixin of *django.views.generic.edit.FormView* with default template and form to upload files and importent content.

**Parameters**

**model** Model where the file will be save

*By default this values is FileHistory*

**template_name** Template name to be used with FormView

*By default is data_importer/data_importer.html*

**form_class** Form that will be used to upload file

*By default this value is FileUploadForm*

**task** Task that will be used to parse file imported

*By default this value is DataImpoterTask*

**importer** Must be one data_importer.importers class that will be used to validate data.

**is_task** Use importer in async mode.

**success_url** Redirect to success page after importer

**extra_context** Set extra context values in template

## 6.2 Usage example

```
class DataImporterCreateView(DataImporterForm):
    extra_context = {'title': 'Create Form Data Importer',
                     'template_file': 'myfile.csv'}
    importer = MyImporterModel
```

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

## d

## C



## D



## E



## G

## T



## U



## X