Django-CryptographicFields

Release 2.1.0

Shahprogrammer

Feb 15, 2021

CONTENTS

1 Installation **3**

2 Settings **5**

3 Fields **7**
3.1 List of Model Fields supported by CryptographicFields: 7

4 Filters **9**
4.1 Sorter function:- 9
4.2 List of sort functions provided by CryptographicFields:- 9
4.3 Order function:- 11

5 Examples **13**
5.1 Cryptography by example 13
5.2 Sorting by example 14
5.3 Ordering by example 14

6 CryptographicFields package **15**
6.1 CryptographicFields.cryptography module 15
6.2 CryptographicFields.fields module 16
6.3 CryptographicFields.filters module 27

7 Indices and tables **33**

Python Module Index **35**

Index **37**

A package for cryptography in Django, wrapping the Python Pycryptodome library.

CHAPTER

ONE

INSTALLATION

```
pip install Django-CryptographicFields
```

Requirements

- Python (3.6+)
- Django (3.0+)
- Pycryptodome (3.9+)
- Timestring (1.6+) Mandatory for python < 3.7

CHAPTER

TWO

SETTINGS

```
INSTALLED_APPS = [
    ...
    'CryptographicFields',
]
```

CRYPTOGRAPHIC_KEY

Default: None

When value of CRYPTOGRAPHIC_KEY is not None a key will be derived from CRYPTOGRAPHIC_KEY. Otherwise the value will be used for the key will be from SECRET_KEY. While specifiing key make sure that it must contain 50 letters otherwise it will raise an error *CryptographicFields.cryptography.LengthError*

CHAPTER

THREE

FIELDS

3.1 List of Model Fields supported by CryptographicFields:

- *CryptographicFields.fields.BigIntegerField*
- *CryptographicFields.fields.BooleanField*
- *CryptographicFields.fields.BinaryField*
- *CryptographicFields.fields.CharField*
- *CryptographicFields.fields.DateField*
- *CryptographicFields.fields.DateTimeField*
- *CryptographicFields.fields.DecimalField*
- *CryptographicFields.fields.EmailField*
- *CryptographicFields.fields.FilePathField*
- *CryptographicFields.fields.FloatField*
- *CryptographicFields.fields.IntegerField*
- *CryptographicFields.fields.GenericIPAddressField*
- *CryptographicFields.fields.PositiveBigIntegerField*
- *CryptographicFields.fields.PositiveIntegerField*
- *CryptographicFields.fields.PositiveSmallIntegerField*
- *CryptographicFields.fields.SlugField*
- *CryptographicFields.fields.SmallIntegerField*
- *CryptographicFields.fields.TextField*
- *CryptographicFields.fields.URLField*
- *CryptographicFields.fields.UUIDField*
- *CryptographicFields.fields.TimeField*

CHAPTER

FOUR

FILTERS

4.1 Sorter function:-

- *CryptographicFields.filters.sort()*

4.2 List of sort functions provided by CryptographicFields:-

- *CryptographicFields.filters.startswith()*
- *CryptographicFields.filters.istartswith()*
- *CryptographicFields.filters.endswith()*
- *CryptographicFields.filters.iendswith()*
- *CryptographicFields.filters.contains()*
- *CryptographicFields.filters.icontains()*
- *CryptographicFields.filters.lt()*
- *CryptographicFields.filters.lte()*
- *CryptographicFields.filters.gt()*
- *CryptographicFields.filters.gte()*
- *CryptographicFields.filters.range()*
- *CryptographicFields.filters.date()*
- *CryptographicFields.filters.date_lt()*
- *CryptographicFields.filters.date_lte()*
- *CryptographicFields.filters.date_gt()*
- *CryptographicFields.filters.date_gte()*
- *CryptographicFields.filters.date_range()*
- *CryptographicFields.filters.year()*
- *CryptographicFields.filters.year_lt()*
- *CryptographicFields.filters.year_lte()*
- *CryptographicFields.filters.year_gt()*
- *CryptographicFields.filters.year_gte()*

- *CryptographicFields.filters.year_range()*
- *CryptographicFields.filters.month()*
- *CryptographicFields.filters.month_lt()*
- *CryptographicFields.filters.month_lte()*
- *CryptographicFields.filters.month_gt()*
- *CryptographicFields.filters.month_gte()*
- *CryptographicFields.filters.month_range()*
- *CryptographicFields.filters.day()*
- *CryptographicFields.filters.day_lt()*
- *CryptographicFields.filters.day_lte()*
- *CryptographicFields.filters.day_gt()*
- *CryptographicFields.filters.day_gte()*
- *CryptographicFields.filters.day_range()*
- *CryptographicFields.filters.time()*
- *CryptographicFields.filters.time_lt()*
- *CryptographicFields.filters.time_lte()*
- *CryptographicFields.filters.time_gt()*
- *CryptographicFields.filters.time_gte()*
- *CryptographicFields.filters.time_range()*
- *CryptographicFields.filters.hour()*
- *CryptographicFields.filters.hour_lt()*
- *CryptographicFields.filters.hour_lte()*
- *CryptographicFields.filters.hour_gt()*
- *CryptographicFields.filters.hour_gte()*
- *CryptographicFields.filters.hour_range()*
- *CryptographicFields.filters.minute()*
- *CryptographicFields.filters.minute_lt()*
- *CryptographicFields.filters.minute_lte()*
- *CryptographicFields.filters.minute_gt()*
- *CryptographicFields.filters.minute_gte()*
- *CryptographicFields.filters.minute_range()*
- *CryptographicFields.filters.second()*
- *CryptographicFields.filters.second_lt()*
- *CryptographicFields.filters.second_lte()*
- *CryptographicFields.filters.second_gt()*
- *CryptographicFields.filters.second_gte()*

- *CryptographicFields.filters.second_range()*
- *CryptographicFields.filters.regex()*

4.3 Order function:-

- *CryptographicFields.filters.order_by()*

CHAPTER

FIVE

EXAMPLES

5.1 Cryptography by example

Using symmetrical encryption to store sensitive data in the database.

```
from django.db import models

from CryptographicFields.fields import *


class User(models.Model):
     uuid = UUIDField()
     username = CharField(max_length=20)
     first_name = CharField(max_length=120)
     last_name = CharField(max_length=120)
     age = SmallIntegerField()
     email = EmailField()
     joined = DateTimeField()
```

The data will now be automatically encrypted when saved to the database & decrypted when retrived with django's ORM.CryptographicFields uses an encryption that allows for bi-directional data retrieval.

Generating Data

```
from .models import User
from uuid import uuid5,NAMESPACE_URL
import datetime

User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.a.random.com"), username=
↪"admin", first_name="Albert",
                                    last_name="Frost", age=24, email="albert@gmail.
↪com", joined=datetime.datetime(2015, 12, 26, 18, 35, 54))
User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.b.random.com"), username=
↪"empi16", first_name="Empi",
                                    last_name="Tsar", age=16, email="empi@rediff.com
↪", joined=datetime.datetime(2025, 12, 28, 12, 35, 34))
User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.c.random.com"), username=
↪"dextEr", first_name="Dexter",
                                    last_name="Flutnes", age=28, email=
↪"dextEr@random.com", joined=datetime.datetime(2018, 2, 20, 20, 50, 15))
User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.d.random.com"), username=
↪"shahprogrammer", first_name="Dhwanil",
                                    last_name="Shah", age=18, email=
↪"shahprogrammer@random.com", joined=datetime.datetime(2018, 4, 1, 20, 25, 14))
```

(continues on next page)

(continued from previous page)

```
User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.e.random.com"), username=
↪"graphin", first_name="Graphin",
                                    last_name="frost", age=30, email="graphin@yahoo.
↪com", joined=datetime.datetime(2005, 9, 22, 9, 33, 40))
User.objects.create(uuid=uuid5(NAMESPACE_URL, "https://www.f.random.com"), username=
↪"abc", first_name="abc",
                                    last_name="xyz", age=16, email="abc@yahoo.com",␣
↪joined=datetime.datetime(2009, 7, 22, 14, 5, 40))
```

5.2 Sorting by example

```
from .models import User
from CryptographicFields.filters import sort,iendswith,startswith

# You can use any method that generate QuerySet object like all,filters etc
queryset=User.objects.all()
"""Sort function takes first arg Queryset second sort_function third field name
fourth your query value & return new QuerySet object with sorted data"""
# using iendswith to sort queryset
sort_queryset=sort(queryset, iendswith, 'username', "er")
print(sort_queryset)
<QuerySet [<User: User object (3)>, <User: User object (4)>]>
# using startswith to sort queryset
sort_queryset=sort(self.queryset, startswith, 'last_name', "F")
print(sort_queryset)
<QuerySet [<User: User object (1)>, <User: User object (3)>]>
```

5.3 Ordering by example

```
from .models import User
from CryptographicFields.filters import order_by

# You can use any method that generate QuerySet object like all,filters etc
queryset=User.objects.all()
"""Order_by functions takes queryset as first arg &
tuple with fields_name from higher priority to lower priority
For eg in this case highest priority is age & lowest priority is username.
i.e it will sort queryset with username field value if it finds two or more objects␣
↪with same age value"""
# Ascending Order
order_queryset=(order_by(self.queryset, ("age", "username"))
print(order_queryset)
<QuerySet [<User: User object (6)>, <User: User object (2)>,
 <User: User object (4)>, <User: User object (1)>,
 <User: User object (3)>, <User: User object (5)>]>
# Descending Order
order_queryset=(order_by(self.queryset, ("age", "username"),reverse=True)
print(order_queryset)
<QuerySet [<User: User object (5)>, <User: User object (3)>,
 <User: User object (1)>, <User: User object (4)>,
 <User: User object (2)>, <User: User object (6)>]>
```

CHAPTER

SIX

CRYPTOGRAPHICFIELDS PACKAGE

6.1 CryptographicFields.cryptography module

exception CryptographicFields.cryptography.**LengthError**(*length*)
Bases: Exception

CryptographicFields.cryptography.**get_key**(*settings*) → str
Gets the encryption for encrypting & decrypting data.

Gets value from CRYPTOGRAPHIC_KEY & if not defined then from SECRET_KEY Checks the len of the key id less than 50 then raise LengthError

Raises ***LengthError*** – It raises when the len of Encryption is less than 50 chars

Returns Key for cryptography

Return type str

CryptographicFields.cryptography.**type_check**(*string*) → bytearray
Checks weather the inputed data is in correct format which is required for encryption & decryption.

Checks weather the inputed data is in correct format which is required for encryption & decryption. Which is in this case is bytearray

Parameters **string** (*Any*) – Data from User

Returns bytes

Return type bytearray

CryptographicFields.cryptography.**to_hex**(*string*) → hex
Converts bytes to hex

Converts the bytes received after encryption to hex for storing it in database

Parameters **string** (*bytes*) – encrypted bytes

Returns hexify the bytes

Return type hex

CryptographicFields.cryptography.**from_hex**(*hexstring*) → bytearray
converts hex to bytearray

Converts the hex string received from databse to bytes for decryption

Parameters **hexstring** (*hex*) – hex string recieved from database

Returns bytes from hex string

Return type bytearray

CryptographicFields.cryptography.**encrypt**(*string*) → hex
Encrypts the data

Encrypts the data recieved from user using AES-256 CFB

Parameters **string** (*Any*) – Data from User

Returns the hex of the encrypted string

Return type hex

CryptographicFields.cryptography.**decrypt**(*hexstring*) → bytearray
Decrypts the data

Decrypts the data recieved from database using AES-256 CFB

Parameters **hexstring** (*hex*) – hex string recieved from database

Returns bytes of decrypted string

Return type bytearray

6.2 CryptographicFields.fields module

class CryptographicFields.fields.**StartsWith**(*lhs*, *rhs*)
Bases: django.db.models.lookups.FieldGetDbPrepValueMixin, django.db.models.lookups.StartsWith

class CryptographicFields.fields.**CharField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**BooleanField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: django.db.models.fields.BooleanField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**DateField**(*verbose_name=None*, *name=None*, *auto_now=False*, *auto_now_add=False*, ***kwargs*)

Bases: django.db.models.fields.DateField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

pre_save(*model_instance*, *add*)
Return field's value just before saving.

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'date': <class 'CryptographicFields.fields.StartsWith'>, 'startswith

class CryptographicFields.fields.**DateTimeField**(*verbose_name=None*, *name=None*, *auto_now=False*, *auto_now_add=False*, ***kwargs*)

Bases: django.db.models.fields.DateTimeField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

pre_save(*model_instance*, *add*)
Return field's value just before saving.

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'date': <class 'CryptographicFields.fields.StartsWith'>, 'startswith

class CryptographicFields.fields.**TimeField**(*verbose_name=None*, *name=None*, *auto_now=False*, *auto_now_add=False*, ***kwargs*)

Bases: django.db.models.fields.TimeField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

pre_save(*model_instance*, *add*)
Return field's value just before saving.

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'time': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**DecimalField**(*verbose_name=None*, *name=None*, *max_digits=None*, *decimal_places=None*, ***kwargs*)

Bases: django.db.models.fields.DecimalField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_save(*value*, *connection*)
Return field's value prepared for saving into a database.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class CryptographicFields.fields.**EmailField**(**args*, ***kwargs*)

Bases: *CryptographicFields.fields.CharField*

default_validators = [<django.core.validators.EmailValidator object>]

description

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**FloatField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: django.db.models.fields.FloatField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class CryptographicFields.fields.**IntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)
Bases: `django.db.models.fields.IntegerField`

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class CryptographicFields.fields.**BigIntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)
Bases: *CryptographicFields.fields.IntegerField*

error_messages

description

MAX_BIGINT = 9223372036854775807

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class CryptographicFields.fields.**GenericIPAddressField**(*verbose_name=None*, *name=None*, *protocol='both'*, *unpack_ipv4=False*, **args*, ***kwargs*)
Bases: django.db.models.fields.GenericIPAddressField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**PositiveBigIntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: django.db.models.fields.PositiveIntegerRelDbTypeMixin, *CryptographicFields.fields.IntegerField*

description

error_messages

to_python(*value: Any*) → Any

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

formfield(***kwargs*)

Return a django.forms.Field instance for this field.

class CryptographicFields.fields.**PositiveIntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: django.db.models.fields.PositiveIntegerRelDbTypeMixin, *CryptographicFields.fields.IntegerField*

description

error_messages

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class CryptographicFields.fields.**PositiveSmallIntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)
Bases: django.db.models.fields.PositiveIntegerRelDbTypeMixin, *CryptographicFields.fields.IntegerField*

description

error_messages

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class CryptographicFields.fields.**SlugField**(**args*, *max_length=50*, *db_index=True*, *allow_unicode=False*, ***kwargs*)
Bases: *CryptographicFields.fields.CharField*

description

default_validators = [<django.core.validators.RegexValidator object>]

deconstruct()
Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.

- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

get_internal_type()

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**SmallIntegerField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: *CryptographicFields.fields.IntegerField*

description

class CryptographicFields.fields.**TextField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: django.db.models.fields.TextField

get_internal_type() → str

to_python(*value: Any*) → Any

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any

Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)

Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)

Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**URLField**(*verbose_name=None*, *name=None*, ***kwargs*)

Bases: *CryptographicFields.fields.CharField*

default_validators = [<django.core.validators.URLValidator object>]

description

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path

- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**BinaryField**(**args*, ***kwargs*)
Bases: django.db.models.fields.BinaryField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**UUIDField**(*verbose_name=None*, ***kwargs*)
Bases: django.db.models.fields.UUIDField

get_internal_type() → str

to_python(*value: Any*) → Any
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any
Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)
Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

clean(*value*, *model_instance*)
Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

class CryptographicFields.fields.**FilePathField**(*verbose_name=None*, *name=None*, *path=''*, *match=None*, *recursive=False*, *allow_files=True*, *allow_folders=False*, ***kwargs*)

Bases: django.db.models.fields.FilePathField

get_internal_type() → str

to_python(*value: Any*) → Any

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

get_prep_value(*value: Any*) → Any

Perform preliminary non-db specific value checks and conversions.

from_db_value(*value: Any*, *expression: Any*, *connection: Any*) → Any

get_db_prep_value(*value*, *connection*, *prepared=False*)

Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

class_lookups = {'startswith': <class 'CryptographicFields.fields.StartsWith'>}

clean(*value*, *model_instance*)

Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

6.3 CryptographicFields.filters module

CryptographicFields.filters.**sort**(*queryset: django.db.models.query.QuerySet*, *lookup_function: Callable[[Union[str, int, float, datetime.datetime, datetime.date, datetime.time, Any], str, Any], bool]*, *field_name: str*, *query: Any*) → django.db.models.query.QuerySet

Filters queryset

Filters queryset as per lookup provided by user &query provided by user

Parameters

- **lookup_function** (*Callable[[Union[str, int, float, datetime, date, time, Any], str, Any], bool]*) – Functions that filters the data
- **queryset** (*QuerySet*) – Queryset generated by querying data
- **field_name** (*str*) – Name of the field which is to be filtered
- **query** (*Any*) – Query for filtering the Queryset

Returns QuerySet with Filtered Data

Return type QuerySet

CryptographicFields.filters.**order_by**(*queryset: django.db.models.query.QuerySet*, *field_name: Tuple[str, ...]*, *reverse: bool = False*) → django.db.models.query.QuerySet

Order Queryset by the given field

Order the Queryset as per field_name given.It supports multiple level of odering

Parameters

- **queryset** (*QuerySet*) – Queryset generated by querying data
- **field_name** (*Tuple*) – Tuple with name of the field from higher priority to lower priority
- **reverse** (*bool, optional*) – Type of ordering (Ascending|Descending), defaults to False

Returns QuerySet with Ordered Data

Return type QuerySet

CryptographicFields.filters.**startswith**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if a string is Starts With the query (Case Sensitive)

CryptographicFields.filters.**istartswith**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if a string is Starts With the query (Not Case Sensitive)

CryptographicFields.filters.**endswith**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if a string is Ends With the query (Case Sensitive)

CryptographicFields.filters.**iendswith**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if a string is Ends With the query (Not Case Sensitive)

CryptographicFields.filters.**contains**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if query is in value of the object (Case Sensitive)

CryptographicFields.filters.**icontains**(*query: str*, *field_name: str*, *object: Any*) → bool
Check if query is in value of the object (Not Case Sensitive)

CryptographicFields.filters.**gt**(*query: Union[int, float]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**gte**(*query: Union[int, float]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**lt**(*query: Union[int, float]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**lte**(*query: Union[int, float]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**date**(*query: Union[datetime.date, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is equal to query

CryptographicFields.filters.**date_lt**(*query: Union[datetime.date, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**date_lte**(*query: Union[datetime.date, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**date_gt**(*query: Union[datetime.date, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**date_gte**(*query: Union[datetime.date, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**date_range**(*query: Tuple[Union[datetime.date, datetime.datetime], Union[datetime.date, datetime.datetime]]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**year**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**year_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**year_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**year_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**year_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**year_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**month**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**month_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**month_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**month_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**month_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**month_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**day**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**day_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**day_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**day_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**day_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**day_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**time**(*query: Union[datetime.time, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**time_lt**(*query: Union[datetime.time, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**time_lte**(*query: Union[datetime.time, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**time_gt**(*query: Union[datetime.time, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**time_gte**(*query: Union[datetime.time, datetime.datetime]*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**time_range**(*query: Tuple[Union[datetime.time, datetime.datetime], Union[datetime.time, datetime.datetime]]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**hour**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**hour_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**hour_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**hour_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**hour_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**hour_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**minute**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**minute_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**minute_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**minute_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**minute_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**minute_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**second**(*query: int*, *field_name: str*, *object: Any*) → bool
Checks if value of object is equal to query

CryptographicFields.filters.**second_lt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than value of query

CryptographicFields.filters.**second_lte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is less than or equal to value of query

CryptographicFields.filters.**second_gt**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than value of query

CryptographicFields.filters.**second_gte**(*query: int*, *field_name: str*, *object: Any*) → bool
Check if value of object is greater than or equal to value of query

CryptographicFields.filters.**second_range**(*query: Tuple[int, int]*, *field_name: str*, *object: Any*) → bool
Check if value of object is in range of query

CryptographicFields.filters.**regex**(*query: Any*, *field_name: str*, *object: Any*) → bool
Checks if query pattern is present in value of object

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

CryptographicFields.cryptography, 15
CryptographicFields.fields, 16
CryptographicFields.filters, 27

INDEX

B

BigIntegerField (*class in CryptographicFields.fields*), 20
BinaryField (*class in CryptographicFields.fields*), 26
BooleanField (*class in CryptographicFields.fields*), 16

C

CharField (*class in CryptographicFields.fields*), 16
class_lookups (*CryptographicFields.fields.BinaryField attribute*), 26
class_lookups (*CryptographicFields.fields.BooleanField attribute*), 17
class_lookups (*CryptographicFields.fields.CharField attribute*), 16
class_lookups (*CryptographicFields.fields.DateField attribute*), 18
class_lookups (*CryptographicFields.fields.DateTimeField attribute*), 18
class_lookups (*CryptographicFields.fields.EmailField attribute*), 19
class_lookups (*CryptographicFields.fields.FilePathField attribute*), 27
class_lookups (*CryptographicFields.fields.GenericIPAddressField attribute*), 21
class_lookups (*CryptographicFields.fields.SlugField attribute*), 24
class_lookups (*CryptographicFields.fields.TextField attribute*), 25
class_lookups (*CryptographicFields.fields.TimeField attribute*), 18
class_lookups (*CryptographicFields.fields.URLField attribute*), 26
class_lookups (*CryptographicFields.fields.UUIDField attribute*), 26
clean() (*CryptographicFields.fields.BinaryField method*), 26
clean() (*CryptographicFields.fields.BooleanField method*), 17
clean() (*CryptographicFields.fields.CharField method*), 16
clean() (*CryptographicFields.fields.DateField method*), 17
clean() (*CryptographicFields.fields.DateTimeField method*), 18
clean() (*CryptographicFields.fields.DecimalField method*), 19
clean() (*CryptographicFields.fields.FilePathField method*), 27
clean() (*CryptographicFields.fields.FloatField method*), 20
clean() (*CryptographicFields.fields.GenericIPAddressField method*), 21
clean() (*CryptographicFields.fields.IntegerField method*), 20
clean() (*CryptographicFields.fields.TextField method*), 25
clean() (*CryptographicFields.fields.TimeField method*), 18
clean() (*CryptographicFields.fields.UUIDField method*), 26
contains() (*in module CryptographicFields.filters*), 28
CryptographicFields.cryptography module, 15
CryptographicFields.fields module, 16
CryptographicFields.filters module, 27

D

date() (*in module CryptographicFields.filters*), 28
date_gt() (*in module CryptographicFields.filters*), 28
date_gte() (*in module CryptographicFields.filters*), 28
date_lt() (*in module CryptographicFields.filters*), 28
date_lte() (*in module CryptographicFields.filters*), 28
date_range() (*in module CryptographicFields.filters*), 28
DateField (*class in CryptographicFields.fields*), 17
DateTimeField (*class in CryptographicFields.fields*),

18
day() (*in module CryptographicFields.filters*), 29
day_gt() (*in module CryptographicFields.filters*), 29
day_gte() (*in module CryptographicFields.filters*), 29
day_lt() (*in module CryptographicFields.filters*), 29
day_lte() (*in module CryptographicFields.filters*), 29
day_range() (*in module CryptographicFields.filters*), 29
DecimalField (*class in CryptographicFields.fields*), 18
deconstruct() (*CryptographicFields.fields.SlugField method*), 23
deconstruct() (*CryptographicFields.fields.URLField method*), 25
decrypt() (*in module CryptographicFields.cryptography*), 16
default_validators (*CryptographicFields.fields.EmailField attribute*), 19
default_validators (*CryptographicFields.fields.SlugField attribute*), 23
default_validators (*CryptographicFields.fields.URLField attribute*), 25
description (*CryptographicFields.fields.BigIntegerField attribute*), 21
description (*CryptographicFields.fields.EmailField attribute*), 19
description (*CryptographicFields.fields.PositiveBigIntegerField attribute*), 22
description (*CryptographicFields.fields.PositiveIntegerField attribute*), 22
description (*CryptographicFields.fields.PositiveSmallIntegerField attribute*), 23
description (*CryptographicFields.fields.SlugField attribute*), 23
description (*CryptographicFields.fields.SmallIntegerField attribute*), 24
description (*CryptographicFields.fields.URLField attribute*), 25

E

EmailField (*class in CryptographicFields.fields*), 19
encrypt() (*in module CryptographicFields.cryptography*), 15
endswith() (*in module CryptographicFields.filters*), 28
error_messages (*CryptographicFields.fields.BigIntegerField attribute*), 20
error_messages (*CryptographicFields.fields.PositiveBigIntegerField attribute*), 22
error_messages (*CryptographicFields.fields.PositiveIntegerField attribute*), 23
error_messages (*CryptographicFields.fields.PositiveSmallIntegerField attribute*), 23

F

FilePathField (*class in CryptographicFields.fields*), 26
FloatField (*class in CryptographicFields.fields*), 19
formfield() (*CryptographicFields.fields.BigIntegerField method*), 21
formfield() (*CryptographicFields.fields.EmailField method*), 19
formfield() (*CryptographicFields.fields.PositiveBigIntegerField method*), 22
formfield() (*CryptographicFields.fields.PositiveIntegerField method*), 23
formfield() (*CryptographicFields.fields.PositiveSmallIntegerField method*), 23
formfield() (*CryptographicFields.fields.SlugField method*), 24
formfield() (*CryptographicFields.fields.URLField method*), 26
from_db_value() (*CryptographicFields.fields.BinaryField method*), 26
from_db_value() (*CryptographicFields.fields.BooleanField method*), 17
from_db_value() (*CryptographicFields.fields.CharField method*), 16
from_db_value() (*CryptographicFields.fields.DateField method*), 17
from_db_value() (*CryptographicFields.fields.DateTimeField method*), 18
from_db_value() (*CryptographicFields.fields.DecimalField method*), 19
from_db_value() (*CryptographicFields.fields.FilePathField method*), 27
from_db_value() (*CryptographicFields.fields.FloatField method*), 19
from_db_value() (*CryptographicFields.fields.GenericIPAddressField method*), 21
from_db_value() (*CryptographicFields.fields.IntegerField method*), 20
from_db_value() (*CryptographicFields.fields.TextField method*), 25
from_db_value() (*CryptographicFields.fields.TimeField method*), 18

from_db_value() (*CryptographicFields.fields.UUIDField method*), 26
from_hex() (*in module CryptographicFields.cryptography*), 15

G

GenericIPAddressField (*class in CryptographicFields.fields*), 21
get_db_prep_save() (*CryptographicFields.fields.DecimalField method*), 19
get_db_prep_value() (*CryptographicFields.fields.BinaryField method*), 26
get_db_prep_value() (*CryptographicFields.fields.BooleanField method*), 17
get_db_prep_value() (*CryptographicFields.fields.CharField method*), 16
get_db_prep_value() (*CryptographicFields.fields.DateField method*), 17
get_db_prep_value() (*CryptographicFields.fields.DateTimeField method*), 18
get_db_prep_value() (*CryptographicFields.fields.FilePathField method*), 27
get_db_prep_value() (*CryptographicFields.fields.FloatField method*), 19
get_db_prep_value() (*CryptographicFields.fields.GenericIPAddressField method*), 21
get_db_prep_value() (*CryptographicFields.fields.IntegerField method*), 20
get_db_prep_value() (*CryptographicFields.fields.TextField method*), 25
get_db_prep_value() (*CryptographicFields.fields.TimeField method*), 18
get_db_prep_value() (*CryptographicFields.fields.UUIDField method*), 26
get_internal_type() (*CryptographicFields.fields.BinaryField method*), 26
get_internal_type() (*CryptographicFields.fields.BooleanField method*), 17
get_internal_type() (*CryptographicFields.fields.CharField method*), 16
get_internal_type() (*CryptographicFields.fields.DateField method*), 17
get_internal_type() (*CryptographicFields.fields.DateTimeField method*), 18
get_internal_type() (*CryptographicFields.fields.DecimalField method*), 19
get_internal_type() (*CryptographicFields.fields.FilePathField method*), 27
get_internal_type() (*CryptographicFields.fields.FloatField method*), 19
get_internal_type() (*CryptographicFields.fields.GenericIPAddressField method*), 21
get_internal_type() (*CryptographicFields.fields.IntegerField method*), 20
get_internal_type() (*CryptographicFields.fields.SlugField method*), 24
get_internal_type() (*CryptographicFields.fields.TextField method*), 25
get_internal_type() (*CryptographicFields.fields.TimeField method*), 18
get_internal_type() (*CryptographicFields.fields.UUIDField method*), 26
get_key() (*in module CryptographicFields.cryptography*), 15
get_prep_value() (*CryptographicFields.fields.BinaryField method*), 26
get_prep_value() (*CryptographicFields.fields.BooleanField method*), 17
get_prep_value() (*CryptographicFields.fields.CharField method*), 16
get_prep_value() (*CryptographicFields.fields.DateField method*), 17
get_prep_value() (*CryptographicFields.fields.DateTimeField method*), 18
get_prep_value() (*CryptographicFields.fields.DecimalField method*), 19
get_prep_value() (*CryptographicFields.fields.FilePathField method*), 27
get_prep_value() (*CryptographicFields.fields.FloatField method*), 19
get_prep_value() (*CryptographicFields.fields.GenericIPAddressField method*), 21
get_prep_value() (*CryptographicFields.fields.IntegerField method*), 20
get_prep_value() (*CryptographicFields.fields.TextField method*), 25
get_prep_value() (*CryptographicFields.fields.TimeField method*), 18
get_prep_value() (*CryptographicFields.fields.UUIDField method*), 26
gt() (*in module CryptographicFields.filters*), 28
gte() (*in module CryptographicFields.filters*), 28

H

hour() (*in module CryptographicFields.filters*), 30
hour_gt() (*in module CryptographicFields.filters*), 30
hour_gte() (*in module CryptographicFields.filters*), 30
hour_lt() (*in module CryptographicFields.filters*), 30
hour_lte() (*in module CryptographicFields.filters*), 30
hour_range() (*in module CryptographicFields.filters*), 30

I

icontains() (*in module CryptographicFields.filters*), 28
iendswith() (*in module CryptographicFields.filters*), 28
IntegerField (*class in CryptographicFields.fields*), 20
istartswith() (*in module CryptographicFields.filters*), 28

L

LengthError, 15
lt() (*in module CryptographicFields.filters*), 28
lte() (*in module CryptographicFields.filters*), 28

M

MAX_BIGINT (*CryptographicFields.fields.BigIntegerField attribute*), 21
minute() (*in module CryptographicFields.filters*), 30
minute_gt() (*in module CryptographicFields.filters*), 30
minute_gte() (*in module CryptographicFields.filters*), 30
minute_lt() (*in module CryptographicFields.filters*), 30
minute_lte() (*in module CryptographicFields.filters*), 30
minute_range() (*in module CryptographicFields.filters*), 30
module
    CryptographicFields.cryptography, 15
    CryptographicFields.fields, 16
    CryptographicFields.filters, 27
month() (*in module CryptographicFields.filters*), 29
month_gt() (*in module CryptographicFields.filters*), 29
month_gte() (*in module CryptographicFields.filters*), 29
month_lt() (*in module CryptographicFields.filters*), 29
month_lte() (*in module CryptographicFields.filters*), 29
month_range() (*in module CryptographicFields.filters*), 29

O

order_by() (*in module CryptographicFields.filters*), 27

P

PositiveBigIntegerField (*class in CryptographicFields.fields*), 21
PositiveIntegerField (*class in CryptographicFields.fields*), 22
PositiveSmallIntegerField (*class in CryptographicFields.fields*), 23
pre_save() (*CryptographicFields.fields.DateField method*), 17
pre_save() (*CryptographicFields.fields.DateTimeField method*), 18
pre_save() (*CryptographicFields.fields.TimeField method*), 18

R

range() (*in module CryptographicFields.filters*), 28
regex() (*in module CryptographicFields.filters*), 31

S

second() (*in module CryptographicFields.filters*), 30
second_gt() (*in module CryptographicFields.filters*), 31
second_gte() (*in module CryptographicFields.filters*), 31
second_lt() (*in module CryptographicFields.filters*), 31
second_lte() (*in module CryptographicFields.filters*), 31
second_range() (*in module CryptographicFields.filters*), 31
SlugField (*class in CryptographicFields.fields*), 23
SmallIntegerField (*class in CryptographicFields.fields*), 24
sort() (*in module CryptographicFields.filters*), 27
StartsWith (*class in CryptographicFields.fields*), 16
startswith() (*in module CryptographicFields.filters*), 28

T

TextField (*class in CryptographicFields.fields*), 24
time() (*in module CryptographicFields.filters*), 29
time_gt() (*in module CryptographicFields.filters*), 30
time_gte() (*in module CryptographicFields.filters*), 30
time_lt() (*in module CryptographicFields.filters*), 30
time_lte() (*in module CryptographicFields.filters*), 30
time_range() (*in module CryptographicFields.filters*), 30
TimeField (*class in CryptographicFields.fields*), 18
to_hex() (*in module CryptographicFields.cryptography*), 15
to_python() (*CryptographicFields.fields.BigIntegerField method*), 21
to_python() (*CryptographicFields.fields.BinaryField method*), 26
to_python() (*CryptographicFields.fields.BooleanField method*), 17

to_python() (*CryptographicFields.fields.CharField method*), 16
to_python() (*CryptographicFields.fields.DateField method*), 17
to_python() (*CryptographicFields.fields.DateTimeField method*), 18
to_python() (*CryptographicFields.fields.DecimalField method*), 19
to_python() (*CryptographicFields.fields.FilePathField method*), 27
to_python() (*CryptographicFields.fields.FloatField method*), 19
to_python() (*CryptographicFields.fields.GenericIPAddressField method*), 21
to_python() (*CryptographicFields.fields.IntegerField method*), 20
to_python() (*CryptographicFields.fields.PositiveBigIntegerField method*), 22
to_python() (*CryptographicFields.fields.PositiveIntegerField method*), 23
to_python() (*CryptographicFields.fields.PositiveSmallIntegerField method*), 23
to_python() (*CryptographicFields.fields.TextField method*), 25
to_python() (*CryptographicFields.fields.TimeField method*), 18
to_python() (*CryptographicFields.fields.UUIDField method*), 26
type_check() (*in module CryptographicFields.cryptography*), 15

U

URLField (*class in CryptographicFields.fields*), 25
UUIDField (*class in CryptographicFields.fields*), 26

Y

year() (*in module CryptographicFields.filters*), 29
year_gt() (*in module CryptographicFields.filters*), 29
year_gte() (*in module CryptographicFields.filters*), 29
year_lt() (*in module CryptographicFields.filters*), 29
year_lte() (*in module CryptographicFields.filters*), 29
year_range() (*in module CryptographicFields.filters*), 29