ginger - Manual

Release 3.0.1537

Josef Hahn

Jan 21, 2021

CONTENTS

CHAPTER

ONE

LICENSE

ginger is written by Josef Hahn under the terms of the AGPLv3.

Please read the *LICENSE* file from the package and the *Dependencies* section for included third-party stuff.

CHAPTER

TWO

ABOUT

Ginger is a news reader for web feeds. Once it is hosted in a web server, a user can login and read her news in the web browser.

Features:

- multi-user web application
- multiple news feeds per user
- message tagging
- powerful message filtering
- very customizable
- scripting interface

It is based on Django and Python and is real-world tested to run in Apache with the wsgi module on Debian Linux.

CHAPTER

THREE

UP-TO-DATE?

Are you currently reading from another source than the homepage? Are you in doubt if that place is up-to-date? If yes, you should visit https://pseudopolis.eu/wiki/pino/projs/ginger and check that. You are currently reading the manual for version 3.0.1537.

CHAPTER

FOUR

MATURITY

ginger is in production-stable state.

CHAPTER

FIVE

DEPENDENCIES

There are external parts that are used by ginger. Many thanks to the projects and all participants.

 *Python 3.7*, required

 *Django 2.2.17*, required

 *Typical GNU/Linux Desktop*, recommended

 *python-feedparser*, required

 *apache 2.4*, recommended : as web server

 *mod_wsgi*, recommended : apache module

 *jquery*, included : licensed under terms of GPLv2.

 *font 'Symbola'*, included : for logo symbol; free for use; copied from here.

 *banner image*, included : _meta/background.png; license public domain; copied from here.

 *all files in /_meta*, included : if not mentioned otherwise, Copyright 2015 Josef Hahn under license CC BY-SA 3.0 license.

CHAPTER

SIX

FIRST STEPS

Please read how to make Ginger ready for the first steps in *Appendix: Installation*.

The following steps require that all preparation steps are finished. In order to login, visit the Ginger root url (e.g. `http://localhost:8000`) in your favorite browser and follow the instructions there.

Depending on your installation procedure, the initial admin user is directly preconfigured, so you can directly login and add some feeds.

6.1 What's Next?

Ginger is a multi-user tool with an own user database.

User configuration must be made with the admin interface. You can create, remove and modify users (reset password, ...) there. The admin user can see it in the menu bar.

Create some users there for your friends/colleagues/clients. Give them your Ginger url and the user data. Let them insert theirs favorite feeds and read the messages there.

Or simply use Ginger with the admin account on a single-user machine :)

CHAPTER

SEVEN

HOW TO USE GINGER

7.1 Screen Structure

When you are logged in, you see the Ginger main application. It always has a top bar with the main menu button. Right after login, you get the default screen, which shows a filter panel on the right side and a message list on the left side. If you use a mobile device, switch between them instead via the main menu.

The main menu also includes the *Feeds* and *Settings* options. While the latter is interesting for experts, the former one is the first step with a new account.

7.2 Feeds

Ginger works by fetching news messages from some web feeds. Those feeds have to comply to the rss or atom (or some more) standard. They are addressed by a http(s) url that you typically can find on the homepage of a feed provider. The feeds dialog is used for adding them to your Ginger account.

7.3 Settings

The settings view allows to customize some aspects of Ginger. Feel free to take a look there. Most things should be easy to understand.

7.4 Default View

The default view is made for browsing all the news that came from your feeds so far. It allows to attach tags to your messages for querying them later. One feed-specific tag is assigned automatically to each message for convenience.

In mobile device mode, you can switch between the list of your messages and between a panel that allows to filter this list (in order to make searches). In large mode, you see both.

7.4.1 Message List

Each message has one box. You can expand them by selecting them. All actions on a message are available from here. You can remove them, mark them as favorite and manage it's tags. You can also open the corresponding web page in a new browser tab.

Some keyboard support is available. You will find some keyboard shortcuts in button tooltips.

7.4.2 Filter Panel

By default, the message list shows all available messages. The filter panel allows to restrict that list by selecting some filter criteria.

You see a tag cloud in the filter panel, which allows you to filter by particular tags. Each checked tag will restrict your list further. Clear filter criteria for returning to a full list.

Store Filters

You can store your complete filter with a meaningful name and load it again in later sessions or set it as your default filter.

CHAPTER

EIGHT

APPENDIX: INSTALLATION

Install Ginger via the installation package for your environment, if a suitable one exists for download. This also takes care of installing dependencies and doing preparation (unless mentioned otherwise in the installation procedure). Use the source code archive as fallback. Please read more details about your installation flavour in the following.

Installation on Debian

Just download the Ginger package for Debian and install it. After the installation, the start menu contains a browser link to an automatically configured (apache2/wsgi-based) Ginger site. Try the *First Steps*.

Installation as Python wheel package

You need a Python environment with enabled 'wheel' functionality. Download the Ginger wheel package and install it. Proceed with *Appendix: Setting Up Ginger*.

Installation From Source

Download the Ginger source package and extract it to a destination. Install the *Dependencies* and proceed with *Appendix: Setting Up Ginger* afterwards.

CHAPTER

NINE

APPENDIX: SETTING UP GINGER

For some platforms, the installation automatically sets up a Ginger service. If so, those are automatically enabled in background on some platforms and to be explicitly started by the user on other ones. The degree of service quality (in regards of performance, security, ...) can also differ between platforms. Read more about your scenario in *Appendix: Installation*.

If your installation package does automatically install a Ginger service and you are fine with this one, you can skip this preparation step, ensure that the service is started, and proceed with *First Steps*.

Otherwise there are many options to set up a Ginger service manually. The easiest way is to use the included personal Ginger server (technically it uses the Django development server). Other ways are to let Ginger run in a full web server. Please find out all existing possibilities in the Django documentation. Real web servers typically provide much higher service quality.

The first step for any installation way is to run the Ginger admin tool. Open a terminal window. You need to have write permissions for the Ginger installation directory for the following steps.

Finding ginger_admin: Now you have to find out the path to `ginger_admin`. For some platforms, the name can be used directly, but on other ones (e.g. if you use the source package) you have to find it somewhere in the Ginger installation directory; typically in `.../_meta/misc/ginger_admin.py` or likewise.

Call this tool this way for setting up a brand new Ginger installation:

```
ginger_admin setup "/home/foo/.ginger"
```

The second parameter specifies a storage location for Ginger runtime data like the database files.

Afterwards you can start the included personal server this way and browse to the printed destination:

```
ginger_admin runserver
```

This way you could - at least for the moment - proceed with *First Steps*.

For public servers you have to run Ginger in a real web server.

The following gives some short hints for installing Ginger in Apache 2.4 with `mod_wsgi`. If you plan to use a different software stack, you should also read the Django documentation. You probably need certain write permissions during the process.

Finding the local settings file: Find the file `settings_local.py`, which is typically stored in the Ginger installation directory or in its `ginger` subdirectory after you set up Ginger.

The `settings_local.py` must be adapted. Adapt the following settings, but don't remove unlisted ones; just ignore them!:

```
DEBUG = False  # at least when everything works
DATABASES = ...  # as it was or with another database
```

(continues on next page)

(continued from previous page)

```
STATIC_ROOT = "/var/lib/ginger/static/"  # used for static files
...
```

Afterwards, run this on a terminal:

```
ginger_admin init
```

Embed Ginger as wsgi application into an Apache2. Use `_meta/misc/apache2.conf` and `_meta/misc/ginger.wsgi` for inspiration.

If you are more comfortable with a different approach to hosting Django applications, find the application in the Ginger installation directory and host it like a typical Django application without doing any `ginger_admin` calls!

CHAPTER

TEN

APPENDIX: SINGLE USER MODE

Ginger can optionally be operated in single user mode without login. It will use the factory default admin account internally and does not provide multi user and authentication functionality anymore.

Add the following line to your `ginger/settings_local.py`:

```
SINGLE_USER_MODE = True
```

CHAPTER

ELEVEN

APPENDIX: USER SCRIPTING

Users are allowed to add custom scripts in the *Settings* dialog. However, for security reasons, backend side scripts are only available to users on a whitelist. This is automatically true in single user mode. Otherwise, set a global configuration with key *allowuserscriptingfor* to a json serialized list of user names for setting up this whitelist.

CHAPTER

TWELVE

APPENDIX: EXTERNAL AUTHENTICATION

Ginger can use an external authentication method instead of the internal user database. Write an executable (e.g. a shell script) with the following behavior:

- Read one line from stdin: this is a user name
- Read one line from stdin: this is the password
- Check that for correctness and print the username if and only if login data are correct
- Terminate

The returned username should be a modified version of the given one if it may contain problematic characters (like `@`, `%`, ...). The easiest solution is to just remove them. The executable must be allowed to be executed by the web server.

Add a line like this in your `ginger/settings_local.py`:

```
EXTERNAL_AUTH_HELPER = '/path/to/your/auth_helper'
```

Note that the admin interface will still use the internal database! Since you don't need it in this case, you should hide it in your server configuration or at least reset the default admin password!

CHAPTER

THIRTEEN

APPENDIX: MANUALLY UPDATING THE DATABASE SCHEME

For installations without a package manager, you have to manually update the database scheme whenever a new version brings some changes in the Ginger data storage structures. Update the source code first and then call this command:

```
ginger_admin update
```

CHAPTER

FOURTEEN

BACKEND API REFERENCE

14.1 ginger package

14.1.1 Subpackages

ginger.main package

Submodules

ginger.main.admin module

Preparations for Django admin panel.

ginger.main.crawler module

Feed crawlers. Used for polling new messages from the web feeds.

class ginger.main.crawler.**Crawler**

Bases: object

For polling messages, instantiate this class and call crawl().

__cleanhtml(*s*, *linktarget*)

Parameters

- **s** (*str*) –
- **linktarget** (*str*) –

Return type str

__garbage_collector(*feed*, *seen_msgs*)

Parameters

- **feed** (ginger.main.models.Feed) –
- **seen_msgs** (*List[*ginger.main.models.NewsMessage*]*) –

Return type str

__translatetag(*s*, *linktarget*, *tree*)

Parameters

- **s** (*str*) –
- **linktarget** (*str*) –
- **tree** (*List*) –

Return type str

crawl()
Polls all feeds for new messages and returns a report string (just interesting for diagnostics).

Return type str

re_attribute = re.compile('[^A-Za-z]([A-Za-z]*)\\s*=\\s*((\'[^\']*\')|(\\"[^\\"]*\\"))

re_elementname = re.compile('[^A-Za-z]*([A-Za-z]*)[^A-Za-z]')

re_slashatbegin = re.compile('<\\s*/')

re_slashatend = re.compile('/\\s*>')

re_tag = re.compile('<[^<>]*>')

ginger.main.messagefilter module

Message filters. Used for restricting the user's message list by particular criteria.

class ginger.main.messagefilter.**AllMessageFilter**
Bases: *ginger.main.messagefilter.MessageFilter*

Matches all messages.

_abc_impl = <_abc_data object>

_tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

class ginger.main.messagefilter.**AndMessageFilter**(*filterstring*)
Bases: *ginger.main.messagefilter.CombinationMessageFilter*

Matches messages by an and-combination of other filters.

_abc_impl = <_abc_data object>

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

class ginger.main.messagefilter.**CombinationMessageFilter**(*filterstring*)
Bases: *ginger.main.messagefilter.MessageFilter*, abc.ABC

Matches messages by a combination of other filters.

_abc_impl = <_abc_data object>

_tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

class ginger.main.messagefilter.**IdMessageFilter**(*id*)
Bases: *ginger.main.messagefilter.MessageFilter*

Matches a message with a particular id.

Parameters **id** (*Union[str, int]*) –

_abc_impl = <_abc_data object>

_tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

class ginger.main.messagefilter.**MessageFilter**
Bases: abc.ABC

Base class for a message filter.

Different instances of different MessageFilter subclasses can be composed together for filter a message source.

_abc_impl = <_abc_data object>

abstract _tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

Return type List[Any]

abstract applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

Parameters

- **elements** (*Iterable[ginger.main.models.NewsMessage]*) –
- **user** (*django.contrib.auth.models.User*) –

Return type List[*ginger.main.models.NewsMessage*]

property native_representation
Returns a serializable representation for this filter.

class ginger.main.messagefilter.**NotMessageFilter**(*filterstring*)
Bases: *ginger.main.messagefilter.MessageFilter*

Matches messages by inverting another filter.

_abc_impl = <_abc_data object>

_tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

class ginger.main.messagefilter.**OrMessageFilter**(*filterstring*)
Bases: *ginger.main.messagefilter.CombinationMessageFilter*

Matches messages by an or-combination of other filters.

_abc_impl = <_abc_data object>

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

class ginger.main.messagefilter.**TagMessageFilter**(*tag*)
Bases: *ginger.main.messagefilter.MessageFilter*

Matches messages by tag.

_abc_impl = <_abc_data object>

_tonativerepresentation()
Returns a serializable representation of the inner data of this filter.

applyfilter(*elements*, *user*)
Returns a list of messages that are matched by this filter from the input list.

applyfilternegated(*elements*, *user*)

ginger.main.messagefilter.**filterbyfilterstring**(*filterstring*)
Creates a *MessageFilter* for a url-encoded filter string.

Parameters **filterstring** (*str*) –

Return type *ginger.main.messagefilter.MessageFilter*

ginger.main.models module

Models.

class ginger.main.models.**Feed**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A web feed.

See ginger.main.crawler for details about how messages are retrieved from feeds.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

_meta = <Options for Feed>

correspondingTags
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

enabled
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_lastFetched(***, *field=<django.db.models.fields.DateTimeField: lastFetched>*, *is_next=True*, ***kwargs*)

get_previous_by_lastFetched(***, *field=<django.db.models.fields.DateTimeField: lastFetched>*, *is_next=False*, ***kwargs*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

lastFetched
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

newsmessage_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

owner
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

owner_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

updateInterval
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

url
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class ginger.main.models.**NewsMessage**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

A message from a web feed.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

__addtag(*tag*, ***, *firescript=True*)

Parameters

- **tag** (*Union[str, Tag]*) –
- **firescript** (*bool*) –

Return type *ginger.main.models.Tag*

__propagate_tags()

Return type None

_meta = <Options for NewsMessage>

add_tag(*tag*)
Adds a tag to this message.

Parameters **tag** (*Union[str, Tag]*) –

Return type None

created
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

delete_later()
Marks a message as deleted.

Return type None

deleted
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

feed
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

feed_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

fetchedAt
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created(*, *field=<django.db.models.fields.DateTimeField: created>*, *is_next=True*, ***kwargs*)

get_previous_by_created(*, *field=<django.db.models.fields.DateTimeField: created>*, *is_next=False*, ***kwargs*)

guid
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

owner
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

owner_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

remove_tag(*tag*)
Removes a tag from this message.

Parameters **tag** (*str*) –

Return type None

save(**args*, ***kwargs*)
Saves the Django model.

Also executes some *ginger.main.userscripting.execute_user_script()*.

seen
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

set_tags(*tags*)
Sets the list of tags for this message.

Parameters **tags** (*Union[List[str], str]*) –

Return type None

summary
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

tags
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

static tagstring_to_taglist(*tags*)
Returns a list of tag names for a space separated string.

Parameters **tags** (*str*) –

Return type List[str]

title
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

url
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class ginger.main.models.**Tag**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A tag. To be assigned to a Message.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

_meta = <Options for Tag>

feed_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

static formatted_name(*name*)
Returns a formatted tag name, cleaned up from forbidden characters.

Parameters **name** (*str*) –

Return type str

static generate_new_tag_name(*name*, *owner*)
Returns a new tag with a name like the given one. If this tag already exists, it will create a new one with a similar name!

Parameters

- **name** (*str*) –
- **owner** (*django.contrib.auth.models.User*) –

Return type *ginger.main.models.Tag*

static get_tag_by_name(*name*, *owner*)
Returns the Tag by name. If it does not yet exist internally, a new one is created.

Parameters

- **name** (*str*) –
- **owner** (*django.contrib.auth.models.User*) –

Return type *ginger.main.models.Tag*

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

newsmessage_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

owner
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

owner_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

tapplyalso
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

tif
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class ginger.main.models.**TagPropagationRule**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A tag propagation rule.

Those rules are automatically processed in background whenever a tag is assigned to a message, and can lead to further tag assignments.

Whenever a message has all of *iftags*, it also gets all of *applyalsotags* assigned.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

_meta = <Options for TagPropagationRule>

applyalsotags
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

iftags
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

owner
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

owner_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class ginger.main.models.**UserConfiguration**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

One user configuration pair of a key and an associated value.

Can store arbitrary json-serializable objects.

Usually each pair is owned by a user. However, in a few cases the backend stores global stuff here as well (with *user=None*).

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

_meta = <Options for UserConfiguration>

configkey
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

configvalue
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get(*user*, *key*, *defaultvalue=None*)
Returns a user configuration value for a user and key.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **key** (*str*) –
- **defaultvalue** (*Optional[Any]*) –

Return type Optional[Any]

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static list(*user*)
Returns the list of all existing keys for a user.

Parameters **user** (*django.contrib.auth.models.User*) –

Return type List[str]

objects = <django.db.models.manager.Manager object>

static remove(*user*, *key*)
Removes a user configuration value for a user and key.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **key** (*str*) –

Return type bool

static set(*user*, *key*, *value*)
Sets a user configuration value for a user and key.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **key** (*str*) –
- **value** (*Optional[Any]*) –

Return type None

user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

user_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ginger.main.projectinformations module

ginger.main.userscripting module

Helpers for user scripting.

ginger.main.userscripting.**execute_user_script**(*user*, *scriptname*, **args*)

Executes a user script.

This is not always enabled and is a noop otherwise. See the documentation for more details.

Parameters

- **user** (*django.contrib.auth.models*) –
- **scriptname** (*str*) –
- **args** (*Any*) –

Return type None

ginger.main.userscripting.**is_execute_user_script_enabled**(*user*)

Returns whether backend side user scripting is available for a given user.

Parameters **user** (*django.contrib.auth.models*) –

Return type bool

ginger.main.views module

Django views.

Most of them are not actually views for user interface presentation but api implementation.

class ginger.main.views.**AccountsActionLoginView**(***kwargs*)
Bases: django.views.generic.base.View

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

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**AccountsActionLogoutView**(***kwargs*)
Bases: django.views.generic.base.View

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

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**FeedView**(***kwargs*)
Bases: django.views.generic.base.View

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

delete(*request*, *feedid*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **feedid** (*int*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**FeedsActionCrawlView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**FeedsView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**FilterView**(***kwargs*)
Bases: django.views.generic.base.View

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

delete(*request*, *filtername*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **filtername** (*str*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**FiltersView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**IndexPageView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Return type django.http.response.HttpResponse

class ginger.main.views.**LoginPageView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**MessageView**(***kwargs*)
Bases: django.views.generic.base.View

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

delete(*request*, *msgid*)

Parameters

- **request** (*django.http.request.HttpRequest*) –

- **msgid** (*int*) –

Return type django.http.response.HttpResponse

patch(*request*, *msgid*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **msgid** (*int*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**MessagesView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**TagPropagationRuleView**(***kwargs*)
Bases: django.views.generic.base.View

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

delete(*request*, *ruleid*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **ruleid** (*int*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**TagPropagationRulesView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

post(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**TagView**(***kwargs*)
Bases: django.views.generic.base.View

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

delete(*request*, *tag*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **tag** (*str*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**TagsView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**UserConfigurationView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*, *key*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **key** (*str*) –

Return type django.http.response.HttpResponse

put(*request*, *key*)

Parameters

- **request** (*django.http.request.HttpRequest*) –
- **key** (*str*) –

Return type django.http.response.HttpResponse

class ginger.main.views.**UserConfigurationsView**(***kwargs*)
Bases: django.views.generic.base.View

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

get(*request*)

Parameters **request** (*django.http.request.HttpRequest*) –

Return type django.http.response.HttpResponse

ginger.main.viewutils module

Some helping hands for ginger.main.views.

ginger.main.viewutils.**_get_default_admin_user**()
Finds or creates the default admin user.

Return type django.contrib.auth.models.User

ginger.main.viewutils.**ensure_ginger_initialized**()
Ensures that Ginger is initialized (e.g. a factory default admin user was created).

Does the initialization on first run. Returns *True* in this case.

Return type bool

ginger.main.viewutils.**login_required**(*fct*)
Decorator for requiring an authenticated user for a view method.

A user who visits such a view without being logged in will be forwarded to the login page and redirected back afterwards.

Parameters **fct** (*Callable*) –

Return type Callable

ginger.main.viewutils.**modellist_to_dictlist**(*models*)
Returns a list of serializable dicts for a list of native models.

Parameters **models** (*List[django.db.models.base.Model]*) –

Return type List[Dict]

ginger.main.viewutils.**render_template**(*templatefile*, *, *request*, *templateargs=None*, *gingercontextargs=None*)
Renders a template with some common data available in its context.

Parameters

- **templatefile** (*str*) –
- **request** (*django.http.request.HttpRequest*) –
- **templateargs** (*Optional[Dict]*) –
- **gingercontextargs** (*Optional[Dict]*) –

Return type str

ginger.main.viewutils.**request_body_object**(*request*)
Deserializes the request body. Typically this is a dictionary of arguments.

Parameters **request** (*django.http.request.HttpRequest*) –

Return type Optional[Any]

Module contents

Ginger application.

14.1.2 Submodules

14.1.3 ginger.settings module

Default Django settings for Ginger.

If you find this file in a ginger installation (instead of the developer's source code), be aware of the fact that this file contains the factory default. Overriding values is possible by writing them to a `settings_local.py` file.

14.1.4 ginger.urls module

Django url mappings for Ginger.

14.1.5 Module contents

Ginger.

14.2 manage module

Ginger management script. For more information, search the web for “django manage.py”.

PYTHON MODULE INDEX

g

m

INDEX

Symbols

A

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U