# Kotti Documentation

***Release 0.5***

**Daniel Nouri**

November 24, 2014

# Contents

Kotti is a light-weight, user-friendly and extensible web content management system. It is licensed under a BSD-like license

CHAPTER 1

# Features

- **User-friendly**: a simple edit interface hides advanced functionality from less experienced users
- **WYSIWYG editor**: includes a rich text editor that lets you edit content like in office applications
- **Security**: advanced user, groups and user roles management; uses access control lists (ACL) to control access to different parts of the site
- **Templating**: extend Kotti with your own look & feel with very little programming required
- **Customizable**: Many aspects of Kotti are configured through a simple INI file
- **Add-ons**: a plug-in system allows third party software to greatly extend Kotti
- **Pluggable authentication**: allows authentication of users through LDAP or other existing user databases
- **Open**: built on top of well-documented, open source components, such as Python, Pyramid and SQLAlchemy
- **Tested**: continuous testing with a test coverage of 100% guarantees Kotti's stability

CHAPTER 2

# Try it out

You can try out Kotti on Kotti's demo site.

CHAPTER 3

# Under the hood

Kotti is written in Python and builds upon on the two excellent libraries Pyramid and SQLAlchemy. Kotti tries to leverage these libraries as much as possible, thus:

- minimizing the amount of code and extra concepts, and
- allowing users familiar with Pyramid and SQLAlchemy to feel right at home since Kotti's API is mostly that of Pyramid and SQLAlchemy.

CHAPTER 4

# Installation

## 4.1 Requirements

- Runs on Python versions 2.6 and 2.7.
- Support for PostgreSQL and SQLite (tested continuously), and a list of other SQL databases (not tested regularly)
- Support for WSGI and a variety of web servers, including Apache

## 4.2 Installation using `virtualenv`

It's recommended to install Kotti inside a virtualenv:

```
virtualenv mysite --no-site-packages
cd mysite
bin/pip install Kotti
```

Kotti uses Paste Deploy for configuration and deployment. An example configuration file is included with Kotti's source distribution:

```
wget https://github.com/Pylons/Kotti/raw/master/development.ini
```

Finally, to run the application under Pyramid 1.3 and better:

```
bin/pserve development.ini
```

Or alternatively, with older versions of Pyramid:

```
bin/pip install PasteScript
bin/paster serve development.ini
```

CHAPTER 5

# Configuration and customization

## 5.1 INI File

Kotti is configured using an INI configuration file. The *Installation* section explains how to get hold of a sample configuration file. The `[app:main]` section in it might look like this:

```
[app:main]
use = egg:Kotti
pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.debug_templates = true
pyramid.default_locale_name = en
pyramid.includes = pyramid_debugtoolbar
                   pyramid_tm
mail.default_sender = yourname@yourhost
sqlalchemy.url = sqlite:///%(here)s/Kotti.db
kotti.site_title = Kotti
kotti.secret = changethis1
```

Various aspects of your site can be changed right here.

## 5.2 Overview of settings

This table provides an overview of available settings. All these settings must go into the `[app:main]` section of your Paste Deploy configuration file.

| Setting | Description |
|---|---|
| **kotti.site_title** | The title of your site |
| **kotti.secret** | Secret token used as initial admin password |
| kotti.secret2 | Secret token used for email password reset token |
| **sqlalchemy.url** | SQLAlchemy database URL |
| **mail.default_sender** | Sender address for outgoing email |
| mail.host | Email host to send from |
| kotti.includes | List of Python configuration hooks |
| kotti.available_types | List of active content types |
| kotti.base_includes | List of base Python configuration hooks |
| kotti.configurators | List of advanced functions for config |
| kotti.root_factory | Override Kotti's default Pyramid *root factory* |
| kotti.populators | List of functions to fill initial database |
| kotti.templates.api | Override `api` used by all templates |
| kotti.asset_overrides | Override Kotti's templates, CSS files and images. |
| kotti.authn_policy_factory | Component used for authentication |
| kotti.authz_policy_factory | Component used for authorization |
| kotti.session_factory | Component used for sessions |
| kotti.date_format | Date format to use, default: `medium` |
| kotti.datetime_format | Datetime format to use, default: `medium` |
| kotti.time_format | Time format to use, default: `medium` |

Only the settings in bold letters required. The rest has defaults.

## 5.3 kotti.secret and kotti.secret2

The value of `kotti.secret` will define the initial password of the `admin` user. This is the initial user that Kotti creates in the user database. So if you put *mysecret* here, use *mysecret* as the password when you log in as `admin`. You may then change the `admin` user's password through the web interface.

The `kotti.secret` token is also used for signing browser session cookies.

The `kotti.secret2` token is used for signing the password reset token.

Here's an example. Make sure you use different values though!

```
kotti.secret = myadminspassword
kotti.secret2 = $2a$12$VVpW/i1MA2wUUIUHwY6v8O
```

## 5.4 Adjusting the look & feel with `kotti.override_assets`

In your settings file, set `kotti.override_assets` to a list of *asset specifications*. This allows you to set up a directory in your package that will mirror Kotti's own and that allows you to override Kotti's templates, CSS files and images on a case by case basis.

As an example, image that we wanted to override Kotti's master layout template. Inside the Kotti source, the layout template is at `kotti/templates/view/master.pt`. To override this, we would add a directory to our own package called `kotti-overrides` and therein put our own version of the template so that the full path to our own custom template is `mypackage/kotti-overrides/templates/view/master.pt`.

We can then register our `kotti-overrides` directory by use of the `kotti.asset_overrides` setting, like so:

```
kotti.asset_overrides = mypackage:kotti-overrides/
```

## 5.5 Using add-ons

Add-ons will usually include in their installation instructions which settings one should modify to activate them. Configuration settings that are used to activate add-ons are:

- `kotti.includes`
- `kotti.available_types`
- `kotti.base_includes`
- `kotti.configurators`

### 5.5.1 kotti.includes

`kotti.includes` defines a list of hooks that will be called by Kotti when it starts up. This gives the opportunity to third party packages to add registrations to the *Pyramid Configurator API* in order to configure views and more.

As an example, we'll add the kotti_twitter extension to add a Twitter profile widget to the right column of all pages. First we install the package from PyPI:

```
bin/pip install kotti_twitter
```

Then we activate the add-on in our site by editing the `kotti.includes` setting in the `[app:main]` section of our INI file. (If a line with `kotti.includes` does not exist, add it.)

```
kotti.includes = kotti_twitter.include_profile_widget
```

kotti_twitter also asks us to configure the Twitter widget itself, so we add some more lines right where we were:

```
kotti_twitter.profile_widget.user = dnouri
kotti_twitter.profile_widget.loop = true
```

The order in which the includes are listed matters. For example, when you add two slots on the right hand side, the order in which you list them here will control the order in which they will appear.

With this configuration, the search widget is displayed on top of the profile widget:

```
kotti.includes =
    kotti_twitter.include_search_widget
    kotti_twitter.include_profile_widget
```

### 5.5.2 kotti.available_types

The `kotti.available_types` setting defines the list of content types available. The default configuration here is:

```
kotti.available_types = kotti.resources.Document kotti.resources.File
```

An example that removes `File` and adds two content types:

```
kotti.available_types =
    kotti.resources.Document
    kotti_calendar.resources.Calendar
    kotti_calendar.resources.Event
```

## 5.6 Configuring authentication and authorization

You can override the authentication and authorization policy that Kotti uses. By default, Kotti uses these factories:

```
kotti.authn_policy_factory = kotti.authtkt_factory
kotti.authz_policy_factory = kotti.acl_factory
```

These settings correspond to pyramid.authentication.AuthTktAuthenticationPolicy and pyramid.authorization.ACLAuthorizationPolicy being used.

## 5.7 Sessions

The `kotti.session_factory` configuration variable allows the overriding of the default session factory. By default, Kotti uses `pyramid_beaker` for sessions.

CHAPTER 6

# Writing add-ons

## 6.1 Screencast

Here's a screencast that guides you through the process of creating a simple Kotti add-on for visitor comments:

## 6.2 Content types

Defining your own content types is easy. The implementation of the Document content type serves as an example here:

```
class Document(Content):
    type_info = Content.type_info.copy(
        name=u'Document',
        add_view=u'add_document',
        addable_to=[u'Document'],
        )

    def __init__(self, body=u"", mime_type='text/html', **kwargs):
        super(Document, self).__init__(**kwargs)
        self.body = body
        self.mime_type = mime_type

documents = Table('documents', metadata,
    Column('id', Integer, ForeignKey('contents.id'), primary_key=True),
    Column('body', UnicodeText()),
    Column('mime_type', String(30)),
)

mapper(Document, documents, inherits=Content, polymorphic_identity='document')
```

You can configure the list of active content types in Kotti by modifying the *kotti.available_types* setting.

### 6.2.1 Using kotti.populators to create your own root object

If you were to totally customize Kotti, and not even include the stock Document type, you would need to follow the template provided by Document, with some attention to detail for configuration and for instantiating a resource hierarchy, especially the root object. For example, let's say that you replace Document with a custom type called Project (updating available types configuration as needed). In your design, under the Project custom type, you might have a hierarchy of other types, the relationships determined by how the type_info.addable_to setup is done, and how the parent property is set for each record on instantiation. When you instantiate the root Project object, the code in the populate() method of resources.py would be something like:

```
root = Project(name="", title="Mother Project", propertyOne="Something", parent=None)
```

NOTE: So, the details are that the root object must have an empty name (name=””) and the parent is None.

## 6.3 Configuring custom views, subscribers and more

*kotti.includes* allows you to hook `includeme` functions that configure your custom views, subscribers and more. An `includeme` function takes the *Pyramid Configurator API* object as its sole argument. An example:

```
def my_view(request):
    from pyramid.response import Response
    return Response('OK')

def includeme(config):
    config.add_view(my_view)
```

By adding the *dotted name string* of your `includeme` function to the *kotti.includes* setting, you ask Kotti to call it on application start-up. An example:

```
kotti.includes = mypackage.views.includeme
```

## 6.4 `kotti.views.slots`

This module allows add-ons to register renderers that add pieces of HTML to the overall page. In other systems, these are called portlets or viewlets.

A simple example that’ll render *Hello, World!* in in the left column of every page:

```
def render_hello(context, request):
    return u'Hello, World!'

from kotti.views.slots import register
from kotti.views.slots import RenderLeftSlot
register(RenderLeftSlot, None, render_hello)
```

Slot renderers may also return `None` to indicate that they don’t want to include anything. We can change our `render_hello` function to include a message only when the context is the root object:

```
from kotti.resources import get_root
def render_hello(context, request):
    if context == get_root():
        return u'Hello, World!'
```

The second argument to `kotti.views.slots.register()` allows you to filter on context. These two are equivalent:

```
from kotti.views.slots import RenderRightSlot
from mypackage.resources import Calendar

def render_agenda1(context, request):
    if isinstance(context, Calendar):
        return '<div>...</div>'
```

```
register(RenderRightSlot, None, render_agenda1)

def render_agenda2(context, request):
    return '<div>...</div>'
register(RenderRightSlot, Calendar, render_agenda2)
```

Usually you'll want to call `kotti.views.slots.register()` inside an `includeme` function and not on a module level, to allow users of your package to include your slot renderers through the `kotti.includes` configuration setting.

## 6.5 `kotti.events`

This module includes a simple events system that allows users to subscribe to specific events, and more particularly to *object events* of specific object types.

To subscribe to any event, write:

```
def all_events_handler(event):
    print event
kotti.events.listeners[object].append(all_events_handler)
```

To subscribe only to *ObjectInsert* events of *Document* types, write:

```
def document_insert_handler(event):
    print event.object, event.request
kotti.events.objectevent_listeners[(ObjectInsert, Document)].append(
    document_insert_handler)
```

Events of type `ObjectEvent` have `object` and `request` attributes. `event.request` may be `None` when no request is available.

Notifying listeners of an event is as simple as calling the `listeners_notify` function:

```
from kotti events import listeners
listeners.notify(MyFunnyEvent())
```

Listeners are generally called in the order in which they are registered.

## 6.6 `kotti.configurators`

Requiring users of your package to set all the configuration settings by hand in the Paste Deploy INI file is not ideal. That's why Kotti includes a configuration variable through which extending packages can set all other INI settings through Python. Here's an example of a function that programmatically modified `kotti.base_includes` and `kotti_principals` which would otherwise be configured by hand in the INI file:

```
# in mypackage/__init__.py
def kotti_configure(config):
    config['kotti.base_includes'] += ' mypackage.views'
    config['kotti.principals'] = 'mypackage.security.principals'
```

And this is how your users would hook it up in their INI file:

```
kotti.configurators = mypackage.kotti_configure
```

## 6.7 Security

Kotti builds mostly on Pyramid's security API and uses its inherited access control lists support. On top of that, Kotti defines *roles* and *groups* support: Users may be collected in groups, and groups may be given roles that define permissions.

The site root's ACL defines the default mapping of roles to their permissions:

```
root.__acl__ == [
    ['Allow', 'system.Everyone', ['view']],
    ['Allow', 'role:viewer', ['view']],
    ['Allow', 'role:editor', ['view', 'add', 'edit']],
    ['Allow', 'role:owner', ['view', 'add', 'edit', 'manage']],
    ]
```

Every Node object has an `__acl__` attribute, allowing the definition of localized row-level security.

The `kotti.security.set_groups()` function allows assigning roles and groups to users in a given context. `kotti.security.list_groups()` allows one to list the groups of a given user. You may also set the list of groups globally on principal objects, which are of type `kotti.security.Principal`.

Kotti delegates adding, deleting and search of user objects to an interface it calls `kotti.security.AbstractPrincipals`. You can configure Kotti to use a different `Principals` implementation by pointing the `kotti.principals_factory` configuration setting to a different factory. The default setting here is:

```
kotti.principals_factory = kotti.security.principals_factory
```

CHAPTER 7

# Cookbook

## 7.1 Internationalization

### 7.1.1 Locale-specific normalization of titles to URLs

Kotti normalizes document titles to URLs by stripping away language specific characters like umlauts or accented characters. This is often undesirable. If you want a locale-specific normalization of titles, you have to configure the package *plone.i18n* which is used by Kotti for the normalization task.

To configure *plone.i18n*, you have to use ZCML. Fortunately, *plone.i18n* comes with normalizers for many different locales, so you often don't have to implement one by yourself. You simply have to activate them by loading *plone.i18n*'s main ZCML file.

ZCML configuration is not supported out of the box, you first have to install the *pyramid_zcml* package. To load *plone.i18n*'s configuration, you also have to install the package *zope.browserresource*.

To install the packages in a `virtualenv`:

```
bin/pip install pyramid_zcml
bin/pip install zope.browserresource
```

Alternatively, list these as dependencies in your package's `setup.py`.

You can then load *plone.i18n*'s configuration via your own ZCML file. For this, create a package which contains a `configure.zcml` like this:

```
<configure xmlns="http://pylonshq.com/pyramid">
  <include package="pyramid_zcml" />
  <include package="zope.browserresource" file="meta.zcml" />
  <include package="zope.browserresource" />
  <include package="plone.i18n" />
</configure>
```

To load your `configure.zcml` on startup, you have to add a startup hook. For example, add a config.py to your package:

```
def includeme(config):
    config.include('pyramid_zcml')
    config.load_zcml('configure.zcml')
```

Setup your locale and the hook with the following settings in your INI file:

```
pyramid.default_locale_name = de
pyramid.includes = mypackage.config.includeme
```

## 7.2 Using Kotti as a library

Kotti may be used as a library instead of as a framework. That is, instead of extending Kotti through hooks such as `kotti.includes`, your application may take full control and use Kotti in a more library-like fashion.

Using this approach it is well possible to use Kotti's individual subsystems in a very customized application that has only little in common with Kotti's own built-in CMS.

Kotti will need some initial set up though for some of its components to be able to work meaningfully. You'll typically call `kotti.base_configure` from your code to take care of that:

```
default_settings = {
    'kotti.authn_policy_factory': 'myapp.authn_policy_factory',
    'kotti.base_includes': (
        'kotti kotti.views kotti.views.login kotti.views.users'),
    'kotti.includes': 'myapp myapp.views',
    'kotti.use_tables': 'orders principals',
    'kotti.populators': 'myapp.resources.populate',
    'kotti.principals_factory': 'myapp.security.Principals',
    'kotti.root_factory': 'myapp.resources.Root',
    'kotti.site_title': 'Myapp',
    }

def main(global_config, **settings):
    settings2 = default_settings.copy()
    settings2.update(settings)
    config = kotti.base_configure(global_config, **settings2)
    return config.make_wsgi_app()
```

The above example configures Kotti so that its user database and security subsystem are set up. Only a handful of tables (`kotti.use_tables`) and a handful of Kotti's views (`kotti.base_includes`) are activated. Furthermore, our application is configured to use a custom root factory (root node) and a custom populator.

In your *PasteDeploy* configuration you'd then wire up your app directly, maybe like this:

```
[app:main]
use = egg:myapp
pyramid.includes = pyramid_tm
mail.default_sender = yourname@yourhost
sqlalchemy.url = sqlite:///%(here)s/myapp.db
kotti.secret = secret
```

CHAPTER 8

# Contact us

Kotti itself is developed on Github. The issue tracker also lives there.

Have a question or a suggestion? Write to Kotti's mailing list or find us on IRC on irc.freenode.net in channel `#kotti`.

CHAPTER 9

# Tests

To run Kotti's automated test suite, do:

```
bin/py.test
```

Or alternatively:

```
bin/python setup.py test
```

You can also run the tests against a different database using the `KOTTI_TEST_DB_STRING` environment variable. By default, Kotti uses an in-memory SQLite database. An example:

```
KOTTI_TEST_DB_STRING=postgresql://kotti:kotti@localhost:5432/kotti-testing bin/python setup.py test
```

**Important**: Never use this feature against a production database. It will destroy your data.

CHAPTER 10

# API

## 10.1 API Documentation

### 10.1.1 `kotti.security`

`kotti.security.`**`set_groups`**(*name*, *context*, *groups_to_set=()*)

Set the list of groups for principal with given `name` and in given `context`.

`kotti.security.`**`list_groups`**(*name*, *context=None*)

List groups for principal with a given `name`.

The optional `context` argument may be passed to check the list of groups in a given context.

**class** `kotti.security.`**`AbstractPrincipals`**

This class serves as documentation and defines what methods are expected from a Principals database.

Principals mostly provides dict-like access to the principal objects in the database. In addition, there's the 'search' method which allows searching users and groups.

'hash_password' is for initial hashing of a clear text password, while 'validate_password' is used by the login to see if the entered password matches the hashed password that's already in the database.

Use the 'kotti.principals' settings variable to override Kotti's default Principals implementation with your own.

**`hash_password`**(*password*)

Return a hash of the given password.

This is what's stored in the database as 'principal.password'.

**`keys`**()

Return a list of principal ids that are in the database.

**`search`**(***kwargs*)

Return an iterable with principal objects that correspond to the search arguments passed in.

This example would return all principals with the id 'bob':

get_principals().search(name=u'bob')

Here, we ask for all principals that have 'bob' in either their 'name' or their 'title'. We pass '*bob*' instead of 'bob' to indicate that we want case-insensitive substring matching:

get_principals().search(name=u'*bob*', title=u'*bob*')

This call should fail with AttributeError unless there's a 'foo' attribute on principal objects that supports search:

get_principals().search(name=u'bob', foo=u'bar')

**validate_password**(*clear*, *hashed*)
Returns True if the clear text password matches the hash.

**class** kotti.security.**Principal**(*name*, *password=None*, *active=True*, *confirm_token=None*, *title=u''*, *email=None*, *groups=()*)
A minimal 'Principal' implementation.

The attributes on this object correspond to what one ought to implement to get full support by the system. You're free to add additional attributes.

- As convenience, when passing 'password' in the initializer, it is hashed using 'get_principals().hash_password'
- The boolean 'active' attribute defines whether a principal may log in. This allows the deactivation of accounts without deleting them.
- The 'confirm_token' attribute is set whenever a user has forgotten their password. This token is used to identify the receiver of the email. This attribute should be set to 'None' once confirmation has succeeded.

## 10.2 Indices and tables

- *genindex*
- *modindex*
- *search*

# Python Module Index

# Index