
Kotti Documentation

Release 0.5

Daniel Nouri

November 24, 2014

1	Kotti CMS	3
2	For Developers	5
3	Installation	7
3.1	Installation	7
4	Configuration	9
4.1	Configuration	9
5	Developer manual	13
5.1	Developer manual	13
6	Cookbook	19
6.1	Close your site for anonymous users	19
6.2	Use a different template for the front page (or any other page)	19
6.3	Internationalization	20
6.4	Using Kotti as a library	21
7	Support and Development	23
8	Automated tests	25
9	Detailed Change History	27
9.1	Change History	27
10	Thanks	31
	Python Module Index	33

Kotti is a high-level, *Pythonic* web application framework. It includes a small and extensible CMS application called the **Kotti CMS**.

Kotti is most useful when you are developing applications that

- have complex security requirements,
- use workflows, and/or
- work with hierarchical data.

Built on top of a number of *best-of-breed* software components, most notably [Pyramid](#) and [SQLAlchemy](#), Kotti introduces only a few concepts itself, thus hopefully keeping the learning curve flat for the developer.

Kotti CMS

You can **try out the built-in CMS** on [Kotti's demo page](#).

The Kotti CMS is a content management system that's heavily inspired by [Plone](#). Its **main features** are:

- **User-friendliness:** editors can edit content where it appears; thus the edit interface is contextual and intuitive
- **WYSIWYG editor:** includes a rich text editor
- **Responsive design:** Kotti builds on [Twitter Bootstrap](#), which looks good both on desktop and mobile
- **Templating:** you can extend the CMS with your own look & feel with almost no programming required (see *[Adjust the look & feel \(kotti.override_assets\)](#)*)
- **Add-ons:** install a variety of add-ons and customize them as well as many aspects of the built-in CMS by use of an INI configuration file (see *[Configuration](#)*)
- **Security:** the advanced user and permissions management is intuitive and scales to fit the requirements of large organizations

For Developers

For developers, Kotti delivers a strong foundation for building different types of web applications that either extend or replace the built-in CMS.

Developers can add and modify through a well-defined API:

- views,
- templates and layout (both via [Pyramid](#)),
- *Content types*,
- portlets (see [kotti.views.slots](#)),
- access control and the user database (see [Security](#)),
- workflows (via [repoze.workflow](#)),
- and much more.

Kotti has a **down-to-earth** API. Developers working with Kotti will most of the time make direct use of the [Pyramid](#) and [SQLAlchemy](#) libraries. Other notable components used but not enforced by Kotti are [Colander](#) and [Deform](#) for forms, and [Chameleon](#) for templating.

[Continuous testing](#) against different versions of Python and with both *PostgreSQL* and *SQLite* and a complete test coverage make Kotti a **stable** platform to work with.

Installation

You can download Kotti from the [Python Package Index](#), it takes only a few moments to install.

3.1 Installation

3.1.1 Requirements

- Runs on Python versions 2.6 and 2.7.
- Support for PostgreSQL and SQLite (tested regularly), and a list of [other SQL databases](#)
- Support for WSGI and a [variety of web servers](#), including Apache

3.1.2 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. Download it:

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

Finally, to run Kotti:

```
bin/pserve app.ini
```

Note: To run the application with older versions of Pyramid, you might need to do instead:

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

Configuration

4.1 Configuration

Contents

- Configuration
 - INI File
 - Overview of settings
 - kotti.secret and kotti.secret2
 - Adjust the look & feel (kotti.override_assets)
 - Use add-ons
 - * kotti.includes
 - * kotti.available_types
 - * kotti.populators
 - Configure the user interface language
 - Configure authentication and authorization
 - Sessions

4.1.1 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.

4.1.2 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 for the 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 <i>root_factory</i>
kotti.populators	List of functions to fill initial database
kotti.asset_overrides	Override Kotti's templates, CSS files and images.
kotti.templates.api	Override <code>api</code> used by all templates
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: <code>medium</code>
kotti.datetime_format	Datetime format to use, default: <code>medium</code>
kotti.time_format	Time format to use, default: <code>medium</code>
pyramid.default_locale_name	Set the user interface language, default <code>en</code>

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

Do take a look at the required settings (in bold) and adjust them in your site's configuration. A few of the settings are less important, and sometimes only used by developers, not integrators.

4.1.3 kotti.secret and kotti.secret2

The value of `kotti.secret` will define the initial password of the admin user. Thus, if you define `kotti.secret = mysecret`, the admin password will be `mysecret`. Log in and change the password at any time 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:

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

Note: Do not use the same values in your site

4.1.4 Adjust the look & feel (`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/
```

4.1.5 Use 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`

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
```

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
```

kotti.populators

The default configuration here is:

```
kotti.populators = kotti.populate.populate
```

Populators are functions with no arguments that get called on system startup. They may then make automatic changes to the database (before calling `transaction.commit()`).

4.1.6 Configure the user interface language

By default, Kotti will display its user interface in English. The default configuration is:

```
pyramid.default_locale_name = en
```

The list of available languages is [here](#).

4.1.7 Configure 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.authn_factory
kotti.authz_policy_factory = kotti.acl_factory
```

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

4.1.8 Sessions

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

Developer manual

5.1 Developer manual

Read the [Configuration](#) section first to understand which hooks both integrators and developers can use to customize and extend Kotti.

Contents

- Developer manual
 - Screencast tutorial
 - Content types
 - Configuring custom views, subscribers and more
 - The Node API
 - `kotti.views.slots`
 - `kotti.events`
 - `kotti.configurators`
 - Security
 - API

5.1.1 Screencast tutorial

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

5.1.2 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.

5.1.3 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 that overrides the default view for all Document content types:

```
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
```

5.1.4 The Node API

One of Kotti's core abstractions is the *Node API*, which is a (*ordered*) *dict-like* API for manipulation of objects and their parent/child relationships.

An example:

```
# Get the root node and set its title:
from kotti.resources import get_root
root = get_root()
root.title = u'A new title'

# Add two pages:
from kotti.resources import Document
root['first'] = Document(title=u'First page')
root['second'] = Document(title=u'Second page')

# Make a copy of the second page and move it into the first page:
root['first']['copy-of-second'] = root['second'].copy()

# Delete the original second page:
del root['first']['second']

# List all children names and nodes:
root.keys()
['first']
```

```
root.values()
[<Document at /first>]
```

5.1.5 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.

5.1.6 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.

5.1.7 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
```

5.1.8 Security

Kotti uses [Pyramid's security API](#), most notably its support [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, which in turn 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
```

5.1.9 API

API Documentation

`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.

Indices and tables

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

6.1 Close your site for anonymous users

This recipe describes how to configure Kotti to require users to log in before they can view any of your site's pages.

To achieve this, we'll have to set our site's ACL. A custom populator will help us do that (see [kotti.populators](#)).

Remember that the default site ACL gives `view` privileges to every user, including anonymous (see [Security](#)). We'll thus have to restrict the `view` permission to the `viewer` role:

```
from transaction import commit
from kotti.resources import get_root

SITE_ACL = [
    (u'Allow', u'role:viewer', [u'view']),
    (u'Allow', u'role:editor', [u'view', u'add', u'edit']),
]

def populate():
    site = get_root()
    site.__acl__ = SITE_ACL
    commit()
```

6.2 Use a different template for the front page (or any other page)

This recipe describes a way to override the template used for a specific object in your database. Imagine you want your front page to stand out from the rest of your site and use a unique layout.

We can set the *default view* for any content object by settings its `default_view` attribute, which is usually `None`. Inside our own populator (see [kotti.populators](#)), we write this:

```
from transaction import commit
from kotti.resources import get_root

def populate():
    site = get_root()
    site.default_view = 'front-page'
    commit()
```

What's left is to register the `front-page` view:

```
def includeme(config):
    config.add_view(
        name='front-page',
        renderer='myapp:templates/front-page.pt',
    )
```

Note: If you want to override instead the template of *all pages*, not only that of a particular page, you should look at the `kotti.override_assets` setting (*Adjust the look & feel (kotti.override_assets)*).

6.3 Internationalization

6.3.1 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
```

6.4 Using Kotti as a library

Instead of taking control of your application, and delegating to your extension, you may use Kotti in applications where you define the main *entry point* yourself.

You'll anyway still need to call `kotti.base_configure` from your code to set up essential parts of Kotti:

```
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
```

Support and Development

Please report any bugs that you find to the [issue tracker](#).

If you've got questions that aren't answered by this documentation, contact the [Kotti mailing list](#) or join the #kotti IRC channel.

Kotti itself is [developed on Github](#). You can check out Kotti's source code via its GitHub repository. Use this command:

```
git clone git@github.com:Pylons/Kotti
```

Automated tests

To run Kotti's automated test suite, do:

```
bin/py.test
```

Or, alternatively:

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

Detailed Change History

9.1 Change History

9.1.1 0.5.1 - 2011-02-27

- Internationalize user interface. Add Portuguese as the first translation.
- A new add menu allows for a more intuitive adding of items in the CMS.
- Refine `Node.copy`. No longer copy over local roles per default.

9.1.2 0.5.0 - 2011-02-15

- Move Kotti's default user interface to use Twitter Bootstrap 2.
- Add a new 'File' content type.
- Add CSRF protection to some forms.
- Remove Kotti's `FormController` in favor of using `pyramid_deform`.
- Use `plone.i18n` to normalize titles to URL parts.
- Add a separate navigation screen that replaces the former intelligent breadcrumbs menu.
- Use `pyramid_beaker` as the default session factory.
- Make `kotti.messages.send_set_password` a bit more flexible.

9.1.3 0.4.5 - 2012-01-19

- Add `'kotti.security.has_permission'` which may be used instead of `'pyramid.security.has_permission'`.

The difference is that Kotti's version will set the "authorization context" to be the context that you pass to `'has_permission'`. The effect is that `'list_groups'` will return a more correct list of local roles, i.e. the groups in the given context instead of `'request.context'`.

- Add a template (`'forbidden.pt'`) for when user is logged in but still getting `HTTPForbidden`.

9.1.4 0.4.4 - 2012-01-05

- The “Forbidden View” will no longer redirect clients that don’t accept ‘text/html’ to the login form.
- Fix bug with ‘kotti.site_title’ setting.

9.1.5 0.4.3 - 2011-12-22

- Add ‘kotti.root_factory’ setting which allows the override Kotti’s default Pyramid *root factory*. Also, make master templates more robust so that a minimal root with ‘__parent__’ and ‘__name__’ can be rendered.
- The ‘kotti.tests’ was factored out. Tests should now import from ‘kotti.testing’.

9.1.6 0.4.2 - 2011-12-20

- More convenient overrides for add-on packages by better use of ‘config.commit()’.

9.1.7 0.4.1 - 2011-12-20

- Modularize Kotti’s Paste App Factory ‘kotti.main’.
- Allow explicit setting of tables that Kotti creates (‘kotti.use_tables’).

9.1.8 0.4.0 - 2011-12-14

- Remove configuration variables ‘kotti.templates.*’ in favour of ‘kotti.asset_overrides’, which uses Pyramid asset specs and their overrides.
- Remove ‘TemplateAPI.__getitem__’ and instead add ‘TemplateAPI.macro’ which has a similar but less ‘special’ API.
- Factor snippets in ‘kotti/templates/snippets.pt’ out into their own templates. Use ‘api.render_template’ to render them instead of macros.

9.1.9 0.3.1 - 2011-12-09

- Add ‘keys’ method to mutation dicts (see 0.3.0).

9.1.10 0.3.0 - 2011-11-30

- Replace *Node.__annotations__* in favor of an extended *Node.annotations*.

Node.annotations will attempt to not only recognize changes to subobjects of type dict, it will also handle list objects transparently. That is, changing arbitrary JSON structures should just work with regard to calling *node.annotations.changed()* when the structure was changed.

9.1.11 0.2.10 - 2011-11-22

- ‘api.format_datetime’ now also accepts a timestamp in addition to datetime.

9.1.12 0.2.9 - 2011-11-21

- Remove MANIFEST.in in favour of using ‘setuptools-git’.

9.1.13 0.2.8 - 2011-11-21

- Remove ‘PasteScript’ dependency since that would result in spurious errors when installing Kotti. See <http://jenkins.danielnouri.org/job/Kotti/42/TOXENV=py27/console>

9.1.14 0.2.7 - 2011-11-20

- Add ‘PasteScript’ dependency.
- Fix #11 where ‘python setup.py test’ would look into a hard-coded ‘bin’ directory.
- Structural analysis documentation. (Unfinished; in ‘analysis’ directory during development. Will be moved to main docs when finished.)

9.1.15 0.2.6 - 2011-11-17

- Add `Node.__annotations__` convenience attribute.
Node.__annotations__ will wrap the annotations dict in such a way that both item and attribute access are possible. It’ll also record changes to dicts inside dicts and mark the parent `__annotations__` attribute as dirty.
- Add a welcome page.
- Delete the demo added in version 0.2.4.

9.1.16 0.2.5 - 2011-11-14

- Add ‘TemplateAPI.render_template’; allow templates to be rendered conveniently from templates.

9.1.17 0.2.4 - 2011-11-13

- Adjust for Pyramid 1.2: INI file, pyramid_tm, Wsgiref server, pcreate and pserve. (MO)
- Add Kotti Demo source and documentation.

9.1.18 0.2.3 - 2011-10-28

- `Node.__getitem__` will now also accept a tuple as key.
`folder['1', '2']` is the same as `folder['1']['2']`, just more efficient.
- Added a new cache decorator based on `repoze.lru`.

9.1.19 0.2.2 - 2011-10-10

- Change the function signature of `kotti.authn_policy_factory`, `kotti.authz_policy_factory` and `kotti.session_factory` to include all settings from the configuration file.

9.1.20 0.2.1 - 2011-09-29

- Minor changes to events setup code to ease usage in tests.

9.1.21 0.2 - 2011-09-16

- No changes.

9.1.22 0.2a2 - 2011-09-05

- Fix templates to be compatible with Chameleon 2. Also, require Chameleon \geq 2.
- Require pyramid \geq 1.2. Also, enable pyramid_debugtoolbar for `development.ini` profile.

9.1.23 0.2a1 - 2011-08-29

- Improve database schema for Nodes. Split `Node` class into `Node` and `Content`.

This change is backward incompatible in that existing content types in your code will need to subclass `Content` instead of `Node`. The example in the docs has been updated. Also, the underlying database schema has changed.

- Improve user database hashing and local roles storage.
- Compatibility fix for Pyramid 1.2.

Thanks

Thanks to the following people for support, code, patches etc:

- Andreas Kaiser
- Andreas Zeidler
- Christian Neumann
- Jeff Pittman
- Mike Orr
- Marco Scheidhuber
- Nuno Teixeira
- Steffen Lindner
- Tom Lazar
- [The University of Coimbra](#)
- [Consipere](#)

k

`kotti.events`, [15](#)

`kotti.views.slots`, [15](#)

A

AbstractPrincipals (class in kotti.security), [17](#)

H

hash_password() (kotti.security.AbstractPrincipals method), [17](#)

K

keys() (kotti.security.AbstractPrincipals method), [17](#)

kotti.events (module), [15](#)

kotti.views.slots (module), [15](#)

L

list_groups() (in module kotti.security), [17](#)

P

Principal (class in kotti.security), [17](#)

S

search() (kotti.security.AbstractPrincipals method), [17](#)

set_groups() (in module kotti.security), [17](#)

V

validate_password() (kotti.security.AbstractPrincipals method), [17](#)