1 Harvest
2 Working with Django models
3 Working with email
4 Porting from Lettuce
5 Installing
6 Using Aloe with Django
7 History
8 Indices and tables
Python Module Index
aloe_django provides utilities to help write Aloe BDD tests for Django applications.
The **harvest** command exposed via Django’s **manage.py** can be used to run Aloe tests under Django with the correct settings.

`harvest` accepts the same flags as `nose` and so these are not extensively documented here.

**<feature>**
- Run only the specified feature files.

**-n N[,N...]**
- Only run the specified scenarios (by number, 1-based) in each feature. Makes sense when only specifying one feature to run, for example:

```
  aloe features/calculator.feature -n 1
```
Working with Django models

Step definitions and utilities for working with Django models.

`aloe_django.steps.models.\texttt{writes_models}(\texttt{model})`

Register a model-specific create and update function.

This can then be accessed via the steps:

| And I have foos in the database: |
| name | bar |
| Baz | Quux |

| And I update existing foos by pk in the database: |
| pk | name |
| 1 | Bar |

A method for a specific model can define a function `write_badgers(data, field)`, which creates and updates the Badger model and decorating it with the `\texttt{writes_models}(\texttt{model_class})` decorator:

```python
@\texttt{writes_models}(Profile)
def write_profile(data, field):
    '''\texttt{Creates a Profile model}'''

    for hash_ in data:
        if field:
            profile = Profile.objects.get(**{field: hash_[field]})
        else:
            profile = Profile()

    ...

    reset_sequence(Profile)
```

The function must accept a list of data hashes and a field name. If field is not None, it is the field that must be used to get the existing objects out of the database to update them; otherwise, new objects must be created for each data hash.

Follow up model creation with a call to `reset_sequence()` to update the database sequences.

If you only want to modify the hash, you can make modifications and then pass it on to `\texttt{write_models}()`.  

```python
@\texttt{writes_models}(Profile)
def write_profile(data, field):
    '''\texttt{Creates a Profile model}'''
```
for hash_ in data:
    # modify hash
    return write_models(Profile, data, field)

aloe_django.steps.models.write_models(model, data, field)

Parameters

• model – a Django model class
• data – a list of hashes to build models from
• field – a field name to match models on, or None

Returns a list of models written

Create or update models for each data hash.

field is the field that is used to get the existing models out of the database to update them; otherwise, if field=None, new models are created.

Useful when registering custom tests with writes_models().

aloe_django.steps.models.tests_existence(model)

Register a model-specific existence test.

This can then be accessed via the steps:

Then foos should be present in the database:
| name | bar |
| badger | baz |

Then foos should not be present in the database:
| name | bar |
| badger | baz |

A method for a specific model can define a function test_badgers(queryset, data) and decorating it with the tests_existence(model_class) decorator:

```python
@tests_existence(Profile)
def test_profile(queryset, data):
    '''Test a Profile model'''

    # modify data ...
    return test_existence(queryset, data)
```

If you only want to modify the hash, you can make modifications then pass it on to test_existence().

aloe_django.steps.models.test_existence(queryset, data)

Parameters

• queryset – a Django queryset
• data – a single model to check for

Returns True if the model exists

Test existence of a given hash in a queryset (or among all model instances if a model is given).

Useful when registering custom tests with tests_existence().
aloem_django.steps.models.reset_sequence(model)
  Reset the ID sequence for a model.

Step (?:Given|And|Then|When) ([A-Z][a-z0-9_]*) with ([a-zA-Z]+) “([a-zA-Z]+)” is linked to ([A-Z][a-z0-9_]*) in the database:
  Link many-to-many models together.

Syntax:
  And model with field “value” is linked to other model in the database:

Example:

<table>
<thead>
<tr>
<th>And</th>
<th>article with name &quot;Guidelines&quot; is linked to tags in the database:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
</tbody>
</table>

Step (?:Given|And|Then|When) ([A-Z][a-z0-9_]*) with ([a-zA-Z]+) “([a-zA-Z]+)” has(?: an?)? ([A-Z][a-z0-9_]*) in the database:
  Create a new model linked to the given model.

Syntax:
  And model with field “value” has new model in the database:

Example:

<table>
<thead>
<tr>
<th>And</th>
<th>project with name &quot;Ball Project&quot; has goals in the database:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>description</td>
</tr>
</tbody>
</table>

Step There should be (d+) ([a-zA-Z][a-z0-9_]*) in the database
  Count the number of models in the database.

Example:

| Then | there should be 0 goals in the database |

Step (?:Given|And|Then|When) (?::an?: )?([A-Z][a-z0-9_]*) should not be present in the database
  Tests for the existence of a model matching the given data.
  Column names are included in a query to the database. To check model attributes that are not database columns
  (i.e. properties). Prepend the column with an @ sign.

Example:

<table>
<thead>
<tr>
<th>Then</th>
<th>foos should not be present in the database:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
</tbody>
</table>

See tests_existence().

Step (?:Given|And|Then|When) (?::an?: )?([A-Z][a-z0-9_]*) should be present in the database
  Test for the existence of a model matching the given data.
  Column names are included in a query to the database. To check model attributes that are not database columns
  (i.e. properties) preprend the column with an @ sign.

Example:

<table>
<thead>
<tr>
<th>Then</th>
<th>foos should be present in the database:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
</tbody>
</table>

See tests_existence().
Step I have an? ([a-z][a-z0-9_]*) in the database:
Create models in the database.

Syntax:

I have model in the database:

Example:

And I have foos in the database:
| name | bar  |
| Baz  | Quux |

See writes_models().

Step I update an? existing ([a-z][a-z0-9_]*) by ([a-z][a-z0-9_]*) in the database:
Update existing models in the database, specifying a column to match on.

Syntax:

I update model by key in the database:

Example:

And I update existing foos by pk in the database:
| pk  | name |
| 1   | Bar  |

See writes_models().
Working with email

Step definitions for working with Django email.

**Step (?:Given|And|Then|When) sending email does not work**
Cause sending email to raise an exception.
This allows simulating email failure.
Example:

```ruby
Given sending email does not work
```

**Step (?:Given|And|Then|When) I clear my email outbox**
Clear the email outbox.
Example:

```ruby
Given I clear my email outbox
```

**Step I have not sent any emails**
Test no emails have been sent.
Example:

```ruby
Then I have not sent any emails
```

**Step (?:And|Then) I have not sent an email with “([^”]*)” in the (subject|body|from_email|to|bcc|cc)**
Test an email does not contain (assert text not in) the given text in the relevant message part (accessible as an attribute on the email object).
This step strictly applies whitespace.
Syntax:
I have not sent an email with “text” in the part
Example:

```ruby
Then I have not sent an email with "pandas" in the body
```

**Step (?:And|Then) I have sent an email with the following HTML alternative:**
Test that an email contains the HTML (assert HTML in) in the multiline as one of its MIME alternatives.
The HTML is normalised by passing through Django's `django.test.html.parse_html()`.
Example:
And I have sent an email with the following HTML alternative:

```html
<p><strong>Name:</strong> Sir Panda</p>
<p><strong>Phone:</strong> 04000000000</p>
<p><strong>Email:</strong> sir.panda@pand.as</p>
```

Step (?:And|Then) I have sent an email with “([^\"]*)” in the (subject|body|from_email|to|bcc|cc)
Test an email contains (assert text in) the given text in the relevant message part (accessible as an attribute on the email object).
This step strictly applies whitespace.
Syntax:

I have sent an email with “text” in the part
Example:

Then I have sent an email with "pandas" in the body

Step (?:And|Then) I have sent an email with the following in the body:
Test the body of an email contains (assert text in) the given multiline string.
This step strictly applies whitespace.
Example:

Then I have sent an email with the following in the body:

```
Dear Mr. Panda,
```

Step (?:And|Then) I have sent (d+) emails?
Test that count mails have been sent.
Syntax:

I have sent count emails
Example:

Then I have sent 2 emails

Chapter 3. Working with email
Porting from Lettuce

The following changes are required to port from Lettuce to aloe_django:

- The decorators `creates_model()` and `checks_existence()` have been removed and should be replaced by `writes_model()` and `tests_existence()` respectively. The prototypes passed to the functions have now been made consistent.

- `hashes_data()` has been removed. Switch to `aloе.tools.guess_types()`.

- Tests are run inside the `aloе_django.TestCase` so a `clean_db()` hook is no longer required.

- The `django_url()` now expects a step as argument. Instead of `django_url(reverse('some-url'))`, you must call `django_url(step, reverse('some-url'))`. `step.test.live_server_url` can also be used to get the root URL of the test server.

- `LETTUCE_USE_TEST_DATABASE` is not supported, the tests are always run using the test database. For a possible speed-up of the test suite, use `--keepdb` option from the Django test runner.

- `LETTUCE_APPS` is not supported. Without any arguments, `harvest` will run all the feature files found in packages in the current directory. To run a subset of tests, specify the features directories as arguments to `harvest`.

- `--debug-mode` is not supported. Use Django’s `settings_override` decorator on the test class to set `DEBUG=True`.


Installing

```
pip install aloe_django
```
Using Aloe with Django

Add `aloe_django` to your project’s `INSTALLED_APPS`.

If you want to run ordinary Python tests using Nose, you should also add `django_nose` to `INSTALLED_APPS` and set the setting `TEST_RUNNER` to `django_nose.NoseTestSuiteRunner`.

```python
GERKIN_TEST_CLASS = 'aloe_django.TestCase'
```

An `aloe.testclass.TestCase` to use to run the tests.

By default this will be `aloe_django.TestCase`, but you can inherit it to change the behaviour of items such as the Django test server (e.g. to enable a threaded server).

See Extending Aloe’s `TestCase` for more details.

```python
GERKIN_TEST_RUNNER = 'aloe_django.runner.GherkinTestRunner'
```

A Nose test runner used when running `manage.py harvest`.

```python
class aloe_django.TestCase
    Base test class for Django Gherkin tests.

    Inherits from both `aloe.testclass.TestCase` and `django.test.LiveServerTestCase`.
```

```python
aloe_django.django_url(step, url=None)
```

The URL for a page from the test server.

**Parameters**

- `step` – A Gherkin step
- `url` – If specified, the relative URL to append.
History

Aloe-Django originally started life as part of the Python BDD tool Lettuce. Like so many succulents, it grew into so much more than that.
CHAPTER 8

Indices and tables

• genindex
• modindex
• search
Python Module Index

a

aloe_django.steps.mail, 9
aloe_django.steps.models, 5
Symbols

-n N[,N...]  
  harvest command line option, 3

A
aloе_django.steps.mail (module), 9
aloе_django.steps.models (module), 5

D
django_url() (in module aloe_django), 15

H
  harvest command line option  
    -n N[N...], 3

R
reset_sequence() (in module aloe_django.steps.models), 6

T
test_existence() (in module aloe_django.steps.models), 6
    TestCase (class in aloe_django), 15
    tests_existence() (in module aloe_django.steps.models), 6

W
write_models() (in module aloe_django.steps.models), 6
writes_models() (in module aloe_django.steps.models), 5