Odoo Unit Testing Guide

Creating tests

To create a test you first need to create a file inside the tests folder called test_what_you_test.py.
To start import common form odoo.test as follows:

```
from odoo.tests import common
```

Of course you can add other imports needed for the test. After the imports you should define a class. The class name doesn't need specific pre- or suffixes. The naming convention for the class-name is CamelCase. For most tests you need to give common.Transactioncase as an argument to the class. The class definition should look something like this:

```
class DummyTest(common.TransactionCase):
```

Inside the class you can define methods with the prefix test_. Then using snake_case you can name the method. These methods are the actual tests. As a parameter for these methods only self is needed. This is added by default when you define the method. The method definition should look something like this:

```
def test_dummy_test(self):
```

Inside this method you can compare and check using self.assert. Some assert-statements are:

- self.assertEqual()
- self.assertTrue()
- self.assertRaises()

After creating your tests you need to update the __init__.py file inside the tests folder. You simply need to add the following line:

```
from . import test_what_you_test
```

Creating meeting_scheduler objects in the tests

There are two ways to create meeting_scheduler objects inside of your tests.

1.

```
meeting_1 = self.env['meeting_scheduler'].create({'meeting_title': 'test meeting',
                                    'meeting_repetitions': 1,
                                    'meeting_frequency': '0',
                                    'meeting_start_date':
                                       datetime.datetime.strptime("2023-03-29 15:00:00", "%Y-%m-%d %H:%M:%S"),
                                    'meeting_end_date':
                                       datetime.datetime.strptime("2023-03-29 17:00:00", "%Y-%m-%d %H:%M:%S"),
                                    'meeting_location': 'airport',
                                    'meeting_subject': 'testing',
                                    'meeting_privacy': 'public',
                                    'meeting_show_as': 'busy'})
```

2.

```
datalist = {'meeting_title': 'meeting name',
         'meeting_repetitions': 1,
         'meeting_frequency': '0',
         'meeting_start_date':datetime.datetime.strptime("2023-03-29 15:00:00", "%Y-%m-%d %H:%M:%S"),
         'meeting_end_date':datetime.datetime.strptime("2023-03-29 17:00:00", "%Y-%m-%d %H:%M:%S"),
         'meeting_location': 'airport',
         'meeting_subject': 'testing',
         'meeting_privacy': 'public',
         'meeting_show_as': 'busy'
         }

meeting_1 = self.env['meeting_scheduler'].create(datalist)
```

Using meeting_scheduler object

After creating a meeting_scheduler object, you can use the funcions defined in the class meeting_scheduler with a dot.
Example:

```
meeting._calc_duration()
```

Internal variables of the meeting_scheduler object can also be accessed with a dot.
Example:

```
meeting.meeting_start_date
```

To find meeting_scheduler objects with certain values you can use the following command:

```
meeting = self.env['meeting_scheduler'].search([('meeting_title', '=', “meeting #1”)])
```

Running Tests

To run the tests you have created you need to make sure that you import the tests in the __init__.py file inside the tests folder. If you do then you need to open a terminal and run one of the following commands:

1) With docker: sudo docker-compose run web --test-enable --stop-after-init -d {database-name} -i meeting_scheduler
2) With odoo locally installed: sudo odoo --test-enable --stop-after-init -d {database-name} -i meeting_scheduler

!!IMPORTANT!!
If you run tests on a locally installed odoo instance you need to make sure that your meeting_scheduler folder is inside the standard addon folder in odoo and not in custom-addons.

Furthermore if you try to run tests for the first time you may get a “OperationalError: FATAL: role “some_name” does not exist”. This error can be fixed by creating a new user in postgres with the name some_name. This can be done by following these steps:

1. sudo su postgres
2. psql
3. create user some_name with password 'new_password' superuser

[Source]

If you ran the tests successfully you should see something like this in your terminal:

```
panda@ubuntu:~$ sudo odoo --test-enable --stop-after-init -d {pandaDB} -i meeting_scheduler
[sudo] password for panda:
Running as user 'root' is a security risk.
2023-03-30 08:09:19,441 7213 INFO ? odoo: Odoo version 14.0-20230309
2023-03-30 08:09:19,442 7213 INFO ? odoo: addons paths: ['/usr/lib/python3/dist-packages/odoo/addons', '/root/.local/share/Odoo/addons/14.0']
2023-03-30 08:09:19,442 7213 INFO ? odoo: database: default@default:default
Warn: Can't find .pfb for face 'Times-Roman'
2023-03-30 08:09:20,458 7213 INFO ? odoo.addons.base.models.ir_actions_report: You need Wkhtmltopdf to print a pdf version of the reports.
2023-03-30 08:09:21,085 7213 INFO ? odoo.service.server: HTTP service (werkzeug) running on ubuntu:8069
2023-03-30 08:09:21,232 7213 INFO {pandaDB} odoo.modules.loading: loading 1 modules...
2023-03-30 08:09:21,289 7213 INFO {pandaDB} odoo.modules.loading: 1 modules loaded in 0.06s, 0 queries (+0 extra)
2023-03-30 08:09:21,639 7213 INFO {pandaDB} odoo.modules.loading: updating modules list
2023-03-30 08:09:21,659 7213 INFO {pandaDB} odoo.addons.base.models.ir_module: ALLOW access to module.update_list on [] to user __system__ #1 via n/a
2023-03-30 08:09:25,858 7213 INFO {pandaDB} odoo.modules.loading: loading 23 modules...
2023-03-30 08:09:26,515 7213 INFO {pandaDB} odoo.modules.loading: Loading module meeting_scheduler (19/23)
2023-03-30 08:09:27,107 7213 INFO {pandaDB} odoo.modules.registry: module meeting_scheduler: creating or updating database tables
2023-03-30 08:09:27,428 7213 INFO {pandaDB} odoo.modules.loading: loading meeting_scheduler/views/views.xml
2023-03-30 08:09:27,577 7213 INFO {pandaDB} odoo.modules.loading: Module meeting_scheduler: loading demo
2023-03-30 08:09:27,618 7213 INFO {pandaDB} odoo.addons.meeting_scheduler.tests.test_dummy_test: Starting DummyTest.test_dummy_test_1 ...
2023-03-30 08:09:27,944 7213 INFO {pandaDB} odoo.addons.meeting_scheduler.tests.test_meeting_duration: Starting DurationTest.test_duration ...
2023-03-30 08:09:28,183 7213 INFO {pandaDB} odoo.modules.loading: Module meeting_scheduler loaded in 1.67s (incl. 0.53s test), 94 queries (+90 test)
2023-03-30 08:09:28,402 7213 INFO {pandaDB} odoo.modules.loading: 23 modules loaded in 2.54s, 94 queries (+90 extra)
2023-03-30 08:09:28,635 7213 WARNING {pandaDB} odoo.modules.loading: The model meeting_scheduler has no access rules, consider adding one. E.g. access_meeting_scheduler,access_meeting_scheduler,model_meeting_scheduler_base.group_user,1,0,0,0
2023-03-30 08:09:29,554 7213 INFO {pandaDB} odoo.modules.loading: Modules loaded.
2023-03-30 08:09:29,566 7213 INFO {pandaDB} odoo.service.server: Starting post tests
2023-03-30 08:09:29,568 7213 INFO {pandaDB} odoo.service.server: 0 post-tests in 0.00s, 0 queries
2023-03-30 08:09:29,568 7213 INFO {pandaDB} odoo.tests.runner: 0 failed, 0 error(s) of 2 tests when loading database '{pandaDB}'
2023-03-30 08:09:29,569 7213 INFO {pandaDB} odoo.service.server: Initiating shutdown
2023-03-30 08:09:29,569 7213 INFO {pandaDB} odoo.service.server: Hit CTRL-C again or send a second signal to force the shutdown.
2023-03-30 08:09:29,639 7213 INFO {pandaDB} odoo.sql_db: ConnectionPool(used=0/count=0/max=64): Closed 2 connections
```

Special Tests:

Test if error is thrown:
To test if a function call throws an error, you can use the following code:

```
with self.assertRaises(IndexError) as context:
    self.env['meeting_scheduler'].create(data_list)
```

Here we expect an IndexError (where we access an index that doesn't exist). To run the .create() without getting an error in the console we need to run it inside the assertRaises(). In the brackets of the assertRaises you need to put the error you are expecting. Otherwise the "unexpected" error won't be handled.
It is important to note that context is an array where all the thrown errors are stored.

Testing with Loops:
The following code snippet is an example of testing code with a for loop. It is important that all statements (especially the assert-statement) are dynamic and not statically hard-coded.

```
for i in range(1, 51):
    name = "weekly #" + str(i)
    meeting = self.env['meeting_scheduler'].search([('meeting_title', '=', name)])

    self.assertNotEqual(meeting.meeting_title, False)
```

This code tests if 50 meeting_scheduler objects exist. The .search() function will never return None, therefore you need to check that a required field (like meeting_title) isn't false. If a field has no value then it is set to False by default.

Example

test_something.py:

```
from odoo.tests import common
import datetime

class TestSomething(common.TransactionCase):

    def test_something(self):
        datalist = {'meeting_title': 'meeting name',
                    'meeting_repetitions': 1,
                    'meeting_frequency': '0',
                    'meeting_start_date':datetime.datetime.strptime("2023-03-29 15:00:00", "%Y-%m-%d %H:%M:%S"),
                    'meeting_end_date':datetime.datetime.strptime("2023-03-29 17:00:00", "%Y-%m-%d %H:%M:%S"),
                    'meeting_location': 'airport',
                    'meeting_subject': 'testing',
                    'meeting_privacy': 'public',
                    'meeting_show_as': 'busy'
                    }

        meeting_1 = self.env['meeting_scheduler'].create(datalist)

        self.assertEqual(meeting_1.meeting_title, "meeting name")
```

__init__.py:

```
from . import test_something
```