odoo-test.sh Documentation

IT-Projects LLC

Dec 26, 2020

Contents:

1 Unit tests **1**
- 1.1 Python Autotests . . . 1
  - 1.1.1 How to run tests . . . 2
  - 1.1.2 Odoo unittest . . . 2
  - 1.1.3 at_install, post_install . . . 5
- 1.2 JS Autotests . . . 5
  - 1.2.1 self.phantom_js() . . . 6
  - 1.2.2 JS tests via Tours . . . 7
  - 1.2.3 How js tour works in Odoo unittests . . . 8
  - 1.2.4 Phantom_js + python tests . . . 9
  - 1.2.5 Screenshots in PhantomJS tests . . . 9
  - 1.2.6 Longpolling in unit tests . . . 9

2 Quality assurance tools **11**
- 2.1 Emulation of slow internet connections in browser . . . 11
  - 2.1.1 Emulation of package lossing . . . 11
- 2.2 Emulation barcode . . . 12
  - 2.2.1 Emulation via OS . . . 12
  - 2.2.2 Emulation via browser . . . 12
- 2.3 ESC/POS printer emulation . . . 13
  - 2.3.1 hw_escpos . . . 13
  - 2.3.2 POS . . . 13
- 2.4 Paypal testing . . . 13
  - 2.4.1 Create developer account . . . 14
  - 2.4.2 Add seller and buyer . . . 14
  - 2.4.3 Configure odoo . . . 14
  - 2.4.4 Directly testing . . . 14

3 JS tour **15**
- 3.1 Tour Definition . . . 16
  - 3.1.1 10.0+ . . . 16
- 3.2 Open backend menu . . . 18
  - 3.2.1 10.0 . . . 18
  - 3.2.2 11.0+ . . . 18
- 3.3 Manual launching . . . 19
  - 3.3.1 10.0+ . . . 19
- 3.4 Auto Launch after installation . . . 19

4 Documentation archive **21**
4.1 Tour Definition . . . 21
4.1.1 8.0, 9.0 . . . 21
4.2 Open backend menu . . . 23
4.2.1 8.0 . . . 23
4.2.2 9.0 . . . 23
4.3 Manual Launching . . . 24
4.3.1 8.0, 9.0 . . . 24
4.4 Auto Launch after installation . . . 24
4.4.1 8.0, 9.0 . . . 24

CHAPTER 1

Unit tests

1.1 Python Autotests

To add tests you need:

- Create folder named **tests**
- Add `__init__.py` file
- Create a file whose name starts with **test_** (put corresponding import to `__init__.py` file from the previous step)
- Add new Class based on one of *test cases*
- Add test methods whose names start with **test_**

> **Warning:** you shall NOT import `tests` in module folder, i.e. do NOT add `from . import tests` to main `__init__.py` file

Example:

```
from odoo.tests.common import TransactionCase

class TestMessage(TransactionCase):
    at_install = True
    post_install = True

    def test_count(self):
        expected_value = self.do_something()
        actual_value = self.get_value()
        self.assertEqual(expected_value, actual_value)

    def do_something(self):
```

(continues on next page)

(continued from previous page)

```
        ...
```

Documentation:

1.1.1 How to run tests

Use following parameters when you start odoo:

- `--test-enable`
- `-d $DB_CONTAINER`
- `-i $MODULE`
- `--workers=0`

js tests

To run tests with phantomjs tests you also need:

- Install phantomjs or use dockers (see below)
- use `--db-filter=.*`

Docker users

You don't need to remove docker container to run test. You can run it in a separate container

- don't worry about name for new container – just use `--rm` arg
- No need to expose ports

So, to run tests with docker:

- use an odoo database which has required modules installed (otherwise it will test all dependencies too)
- OPTIONAL: stop main odoo container, but keep db container
- run new container, e.g.:

```
docker run --rm --link $DB_CONTAINER:db \
-v /something/at/host:/something/at/container \
itprojectsllc/install-odoo:$ODOO_BRANCH-dev \
-- \
--test-enable \
--workers=0 \
--stop-after-init
-d $DATABASE_NAME \
-i $MODULE
```

1.1.2 Odoo unittest

- *Test classes*
  - *Odoo 15.0+*
  - *Odoo 14.0-*
- *setUp and other methods*
- *Assert Methods*

Test classes

Odoo 15.0+

Since Odoo 15, `SavepointCase` is replaced with updated `TransactionCase`.

Complete list of the classes:

```
class BaseCase(unittest.TestCase):
    """
    Subclass of TestCase for common OpenERP-specific code.

    This class is abstract and expects self.registry, self.cr and self.uid to be
    initialized by subclasses.
    """

class TransactionCase(BaseCase):
    """ Test class in which all test methods are run in a single transaction,
    but each test method is run in a sub-transaction managed by a savepoint.
    The transaction's cursor is always closed without committing.
    The data setup common to all methods should be done in the class method
    `setUpClass`, so that it is done once for all test methods. This is useful
    for test cases containing fast tests but with significant database setup
    common to all cases (complex in-db test data).
    After being run, each test method cleans up the record cache and the
    registry cache. However, there is no cleanup of the registry models and
    fields. If a test modifies the registry (custom models and/or fields), it
    should prepare the necessary cleanup (`self.registry.reset_changes()`).
    """
class SingleTransactionCase(BaseCase):
    """ TestCase in which all test methods are run in the same transaction,
    the transaction is started with the first test method and rolled back at
    the end of the last.
    """

class HttpCase(TransactionCase):
    """ Transactional HTTP TestCase with url_open and phantomjs helpers.
    """
```

Odoo 14.0-

From odoo/tests/common.py:

```
class BaseCase(unittest.TestCase):
    """
```

(continues on next page)

(continued from previous page)

```
    Subclass of TestCase for common OpenERP-specific code.

    This class is abstract and expects self.registry, self.cr and self.uid to be
    initialized by subclasses.
    """

class TransactionCase(BaseCase):
    """ TestCase in which each test method is run in its own transaction,
    and with its own cursor. The transaction is rolled back and the cursor
    is closed after each test.
    """

class SingleTransactionCase(BaseCase):
    """ TestCase in which all test methods are run in the same transaction,
    the transaction is started with the first test method and rolled back at
    the end of the last.
    """

class SavepointCase(SingleTransactionCase):
    """ Similar to :class:`SingleTransactionCase` in that all test methods
    are run in a single transaction *but* each test case is run inside a
    rollbacked savepoint (sub-transaction).

    Useful for test cases containing fast tests but with significant database
    setup common to all cases (complex in-db test data): :meth:`~.setUpClass`
    can be used to generate db test data once, then all test cases use the
    same data without influencing one another but without having to recreate
    the test data either.
    """

class HttpCase(TransactionCase):
    """ Transactional HTTP TestCase with url_open and phantomjs helpers.
    """
```

setUp and other methods

For more information see https://docs.python.org/2.7/library/unittest.html#test-cases

- `setUp()` – Method called to prepare the test fixture. This is called immediately before calling the test method. It's recommended to use in `TransactionCase` and `HttpCase` classes
- `setUpClass()` – A class method called before tests in an individual class run. setUpClass is called with the class as the only argument and must be decorated as a `classmethod()`. It's recommended to use in `SingleTransactionCase` and `SavepointCase` classes

```
@classmethod
def setUpClass(cls):
    ...
```

- `tearDown()`, `tearDownClass` – are called *after* test(s). Usually are not used in odoo tests

Assert Methods

Main methods:

Method	Checks that
assertEqual(a, b)	`a == b`
assertNotEqual(a, b)	`a != b`
assertTrue(x)	`bool(x) is True`
assertFalse(x)	`bool(x) is False`
assertIs(a, b)	`a is b`
assertIsNot(a, b)	`a is not b`
assertIsNone(x)	`x is None`
assertIsNotNone(x)	`x is not None`
assertIn(a, b)	`a in b`
assertNotIn(a, b)	`a not in b`
assertIsInstance(a, b)	`isinstance(a, b)`
assertNotIsInstance(a, b)	`not isinstance(a, b)`

Also, to check error raising:

```
with self.assertRaises(ValidationError):
    # some code that supposed to raise error
    ...
```

1.1.3 at_install, post_install

By default, odoo runs test with paramaters:

```
at_install = True
post_install = False
```

at_install

- runs tests right after loading module's files. It runs only in demo mode.
- runs as if other not loaded yet modules are not installed at all
- runs before marking module as installed, which also leads to not loading module's qweb without fixing it manually (don't forget to use *special environment* in odoo before version 12) .

post_install

- runs after installing all modules in current installation set
- runs after calling `registry.setup_models(cr)`
- runs after calling `model._register_hook(cr)`

1.2 JS Autotests

For automatic web tests odoo uses phantomjs.

How to write automatic js tests:

- Follow instruction for *python tests*

- If you have to make several steps in UI to test something:
  - Create *tour*
  - *Run tour via self.phantom_js()*
- If just one step is enough:
  - Run you js code via *self.phantom_js()*

Documentation:

1.2.1 self.phantom_js()

From odoo/tests/common.py:

```
def phantom_js(self, url_path, code, ready="window", login=None, timeout=60, **kw):
    """ Test js code running in the browser
    - optionnally log as 'login'
    - load page given by url_path
    - wait for ready object to be available
    - eval(code) inside the page
    To signal success test do:
    console.log('ok')
    To signal failure do:
    console.log('error')
    If neither are done before timeout test fails.
    """
```

i.e.

- odoo first loads `url_path` as user `login` (e.g. `'admin'`, `'demo'` etc.) or as non-authed user
- then waits for `ready` condition, i.e. when some js variable (e.g. `window`) become truthy
- then executes js `code`
- then wait for one of condition:
  - someone prints `console.log('ok')` – test passed
  - someone prints `console.log('error')` – test failed
  - `timeout` seconds are passed – test failed

Example

Example from mail_sent:

```
# -*- coding: utf-8 -*-
import odoo.tests


@odoo.tests.common.at_install(False)
@odoo.tests.common.post_install(True)
class TestUi(odoo.tests.HttpCase):

    def test_01_mail_sent(self):
        # wait till page loaded and then click and wait again
        code = """
```

(continues on next page)

(continued from previous page)

```
            setTimeout(function () {
                $(".mail_sent").click();
                setTimeout(function () {console.log('ok');}, 3000);
            }, 1000);
        """
        link = '/web#action=%s' % self.ref('mail.mail_channel_action_client_chat')
        self.phantom_js(link, code, "odoo.__DEBUG__.services['mail_sent.sent'].is_
↪ready", login="demo")
```

In this test:

- odoo first loads `/web#action=...` page
- then waits for `odoo.__DEBUG__.services['mail_sent.sent'].is_ready`
  - `odoo.__DEBUG__.services['mail_sent.sent']` is similar to `require('mail_sent.sent')`
  - `is_ready` is a variable in sent.js
- then executes js `code`:

  ```
  setTimeout(function () {
      $(".mail_sent").click();
      setTimeout(function () {console.log('ok');}, 3000);
  }, 1000);
  ```

  which clicks on `Sent` menu and gives to the page 3 seconds to load it.

  This code neither throws errors (e.g. via `throw new Error('Some error description')` nor log `console.log('error')`, but you can add ones to your code to catch failed cases you need.
- then if everything is ok, odoo get message `console.log('ok')`

1.2.2 JS tests via Tours

How to run *odoo tours* in *phantom_js* method?

10.0+

```
from odoo.tests.common import HttpCase

class CLASS_NAME(HttpCase):
    def test_NAME(self):

        tour = 'TOUR_NAME'
        self.phantom_js(
            URL_PATH,

            "odoo.__DEBUG__.services['web_tour.tour']"
            ".run('%s')" % tour,

            "odoo.__DEBUG__.services['web_tour.tour']"
            ".tours['%s'].ready" % tour,

            login=LOGIN_OR_NONE
        )
```

8.0, 9.0

```
class CLASS_NAME(...):
    def test_NAME(self):

        self.phantom_js(
            URL_PATH,

            "odoo.__DEBUG__.services['web.Tour']"
            ".run('TOUR_NAME', 'test')",

            "odoo.__DEBUG__.services['web.Tour']"
            ".tours.TOUR_NAME",

            login=LOGIN_OR_NONE
        )
```

1.2.3 How js tour works in Odoo unittests

The order is as following:

- OPEN *url_path* from **python** *phantom_js* method
- WAIT *ready* condition (Truthy or Falsy) from **python** *phantom_js* method
- OPEN *url* from *tour*'s options in **js** file
- WAIT *wait_for* (deferred object) from *tour*'s options in **js** file
- DO first step from **js** *tour*
  - WAIT when *trigger* becomes visible
  - WAIT when *extra_trigger* becomes visible (if *extra_trigger* is presented)
  - EXECUTE action (*run* or click on *trigger*)
- DO NEXT step
  - ...
- STOP Running when:
  - error happens:
    - thrown via `raise`
    - reported via `console.error(...)`
    - reported by tour system on **timeout** for initial *ready* condition. Timeout value is 60 sec and it cannot be changed.
    - reported by tour system on step **timeout**.
    - Odoo 12 and below: reported via `console.log('error', ...)`
  - `'test successful'` is reported via `console.log` (in Odoo 12 and below it was just `'ok'`)
    - directly by code
    - indirectly by tour system when all steps are done
  - **timeout** from **python** *phantom_js* method is occured. Default is 60 sec

1.2.4 Phantom_js + python tests

Odoo 12.0+

Since Odoo 12.0 there is no any problem with mixing calling phantom_js and python code

Odoo 11.0-

If you need you run some python code before or after calling `phantom_js` you shall not use `self.env` and you need to create new env instead:

```
phantom_env = api.Environment(self.registry.test_cr, self.uid, {})
```

This is because `HttpCase` uses special cursor and using regular cursor via `self.env` leads to deadlocks or different values in database.

1.2.5 Screenshots in PhantomJS tests

Open file `odoo/tests/phantomtest.js` and after the line

```
console.log("PhantomTest.run: execution launched, waiting for console.log('ok')...");
```

add following

```
i=1;
setInterval(function(){
    self.page.render('/tmp/phantomjs-'+i+'.png');
    i++;
}, 1000);
```

It will create screenshot every 1 second (you can update it if needed)

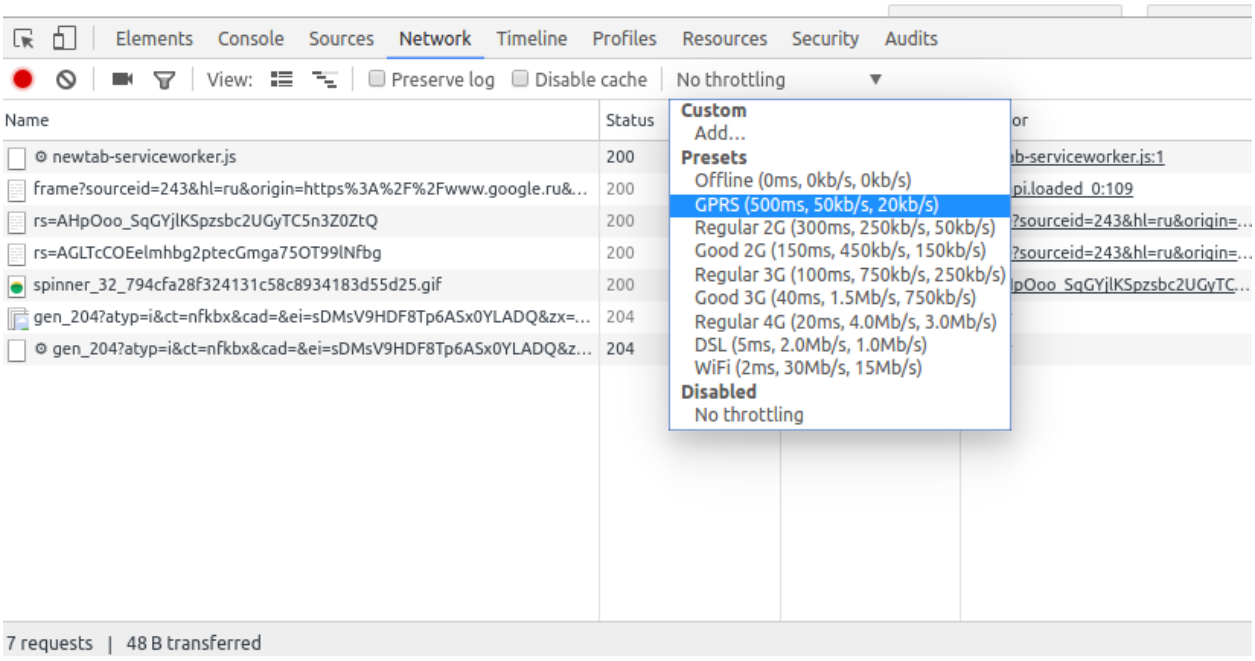
1.2.6 Longpolling in unit tests

It's not possible.

CHAPTER 2

Quality assurance tools

2.1 Emulation of slow internet connections in browser



2.1.1 Emulation of package lossing

In case if you need to emulate *bad* connection, i.e. it works and probably fast, but lose some percents of TCP packages, then do as following

```
# check your network interfaces
ifconfig

# Example below is for eth0
# Other possbile values are
# * wlan0 - wireless connection
# * lo - local connection. Use this, if your run a server on your machine

# lose 30 %
sudo tc qdisc add dev eth0 root netem loss 30%

# "burst of losing"
# Probabilyt of each next lossing depends on previous result.
# For example below:
# Pnext = 0.1 * Pprev + (1-0.1)* Random(0,1)
# Then the package is lost, if Pnext < 0.3
sudo tc qdisc add dev eth0 root netem loss 30% 10%

# show current settings
tc -s qdisc show dev eth0

# reset settings
sudo tc qdisc del dev eth0 root
```

2.2 Emulation barcode

Barcode scanner connected with computer work as keyboard. E.g. after scanning send sequence of symbols as if fast typing on the keyboard.

2.2.1 Emulation via OS

Install **xdotool** app if you haven't it yet.

```
sudo apt-get install xdotool
```

Emulation scanning barcode:

```
sleep 3 && xdotool type 1234567890128 &
```

or so:

```
sleep 3 && xdotool type 3333333333338 &
```

Where: 3 - sleep seconds; 3333333333338 - barcode.

After successfully scanning you will see '3333333333338' in the command line. If toggle to other window that symbols appear in the input field in the this window. So we can send sequence in the app as if we scanning it.

2.2.2 Emulation via browser

Open browser console (e.g. via F12 button) and type (this doesn't work for form inputs):

```
odoo.__DEBUG__.services['web.core'].bus.trigger('barcode_scanned', '1234567890128', $(
↪'.web_client')[0])
```

2.3 ESC/POS printer emulation

2.3.1 hw_escpos

- apply patch

```
cd /path/to/odoo/

# odoo 10
curl https://raw.githubusercontent.com/itpp-labs/odoo-test-docs/master/tools/hw_
↪escpos-patch/hw_escpos-10.patch > hw_escpos.patch

# odoo 9
curl https://raw.githubusercontent.com/itpp-labs/odoo-test-docs/master/tools/
↪master/docs/debugging/hw_escpos-patch/hw_escpos-9.patch > hw_escpos.patch

git apply hw_escpos.patch
```

- install hw_escpos on odoo
- run a separate odoo with following args:

```
-d DB_WITH_HW_ESCPOS --db-filter=DB_WITH_HW_ESCPOS --xmlrpc-port=8888 --workers=0
```

- in new terminal run

```
tail -f /tmp/printer
```

On printing:

- some binary data is sent to /tmp/printer
- odoo prints logs with unparsed data

2.3.2 POS

At any database (including one on runbot as well as database where you have installed hw_escpos):

- set `Receipt printer` checkbox in pos.config and set ip equal to `127.0.0.1:8888`
- open POS interface

> **Warning:** for some reason printer emulation doesn't work in debug mode

- print ticket

2.4 Paypal testing

To test paypal payments you need to:

- Create developer account
- Add seller and buyer in developer sandbox
- Configure odoo
- Directly testing

2.4.1 Create developer account

Go to https://developer.paypal.com/ and create new account.

2.4.2 Add seller and buyer

- Go to **Dashboard->Sand box->Accounts**. Create business (seller) and personal (buyer) accounts. It's recommended to don't use non-ascii symbols in account information (address, name etc.)
- Add some money to buyer (type amount in according field).
- Go to http://sandbox.paypal.com and login as seller. May be you will be forced to apply unconfirmed ssl certificate.
- Follow odoo docs: https://www.odoo.com/documentation/user/14.0/general/payment_acquirers/paypal.html

2.4.3 Configure odoo

- Install **payment_paypal** module
- Go to **Settings->Payments->Payments->Paypal**.
- Pres **Edit**.
- Enter here **Paypal Email ID** - it is *seller* account.
- Follow odoo docs: https://www.odoo.com/documentation/user/14.0/general/payment_acquirers/paypal.html

2.4.4 Directly testing

Open web shop. Buy some goods and pay with paypal. When you will be redirected on paypal page use *buyer* login and password.

CHAPTER 3

JS tour

Tour is a set of steps of possible scenario of module usage.

Steps may be executed automatically for *testing* purpose or by user for *demostrating* purpose.

- *Tour Definition*
  - *10.0+*
    - *Example*
    - *Options*
    - *Step*
    - *Predefined steps*
    - *More documentation*
- *Open backend menu*
  - *10.0*
    - *Manifest*
    - *load_xmlid*
    - *Tour*
  - *11.0+*
- *Manual launching*
  - *10.0+*
- *Auto Launch after installation*

3.1 Tour Definition

3.1.1 10.0+

Example

Example from website_sale module:

```
odoo.define('website_sale.tour', function (require) {
'use strict';

var tour = require("web_tour.tour");
var base = require("web_editor.base");

var options = {
    test: true,
    url: '/shop',
    wait_for: base.ready()
};

var tour_name = 'shop_buy_product';
tour.register(tour_name, options,
    [
        {
            content: "search ipod",
            trigger: 'form input[name="search"]',
            run: "text ipod",
        },
        {
            content: "search ipod",
            trigger: 'form:has(input[name="search"]) .oe_search_button',
        },
        {
            content: "select ipod",
            trigger: '.oe_product_cart a:contains("iPod")',
        },
        {
            content: "select ipod 32GB",
            extra_trigger: '#product_detail',
            trigger: 'label:contains(32 GB) input',
        },
        {
            content: "click on add to cart",
            extra_trigger: 'label:contains(32 GB) input:propChecked',
            trigger: '#product_detail form[action^="/shop/cart/update"] .btn',
        },
        /* ... */
    ]
);

});
```

Options

Options (second argument of `tour.register`):

- **test** – only for tests
- **url** – open link before running the tour
- **wait_for** – wait for deffered object before running the script
- **skip_enabled** – adds *Skip* button in tips

Step

Each step may have following attrubutes:

- **content** – name or title of the step
- **trigger** (mandatory) – where to place tip. *In js tests: where to click*
- **extra_trigger** – when this becomes visible, the tip is appeared. *In js tests: when to click*
- **timeout** – max time to wait for conditions
- **position** – how to show tip (left, rigth, top, bottom), default right
- **width** – width in px of the tip when opened, default 270
- **edition** – specify to execute in *"community"* or in *"enterprise"* only. By default empty – execute at any edition.
- **run** – what to do when tour runs automatically (e.g. in tests)
  - `'text SOMETEXT'` – writes value in **trigger** element
  - `'click'`
  - `'drag_and_drop TO_SELECTOR'`
  - `'auto'` – auto action (click or text)
  - `function: (actions) { ... }` – actions is instance of RunningTourActionHelper – see tour_manager.js for its methods.
- **auto** – step is skipped in non-auto running

Predefined steps

- `tour.STEPS.MENU_MORE` – clicks on menu *More* in backend when visible
- `tour.STEPS.TOGGLE_APPSWITCHER` – nagivate to Apps page when running in enterprise
- `tour.STEPS.WEBSITE_NEW_PAGE` – clicks create new page button in frontend

More documentation

- https://www.odoo.com/slides/slide/the-new-way-to-develop-automated-tests-beautiful-tours-440
- https://github.com/odoo/odoo/blob/10.0/addons/web_tour/static/src/js/tour_manager.js
- https://github.com/odoo/odoo/blob/10.0/addons/web_tour/static/src/js/tip.js

3.2 Open backend menu

3.2.1 10.0

Some additional actions are required to work with backend menus in tours

Manifest

Add `web_tour` to dependencies

```
"depends": [
    "web_tour",
],
# ...
"demo": [
    "views/assets_demo.xml",
    "views/tour_views.xml",
],
```

load_xmlid

You need to set `load_xmlid` for *each* menu you need to open. Recommended name for the file is `tour_views.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Make the xmlid of menus required by the tour available in webclient -->
    <record id="base.menu_administration" model="ir.ui.menu">
        <field name="load_xmlid" eval="True"/>
    </record>
</odoo>
```

Tour

Use *trigger* selector for both editions:

```
{
    trigger: '.o_app[data-menu-xmlid="base.menu_administration"], .oe_menu_
↪toggler[data-menu-xmlid="base.menu_administration"]',
    content: _t("Configuration options are available in the Settings app."),
    position: "bottom"
}
```

3.2.2 11.0+

No additional actions are required.

3.3 Manual launching

3.3.1 10.0+

- activate developer mode.
- Click *Bug* icon (between chat *icon* and *Username* at top right-hand corner)
  - click `Start tour`
- Click *Play* button – it starts tour in auto mode

To run *test-only* tours (or to run tours in auto mode but with some delay) do as following:

- open browser console (F12 in Chrome)
- Type in console:

```
odoo.__DEBUG__.services['web_tour.tour'].run('TOUR_NAME', 1000); // 1000 is delay␣
↪in ms before auto action
```

3.4 Auto Launch after installation

Note: The section archived and now is available *here*.

CHAPTER 4

Documentation archive

Information contained in this section covers **Odoo 8 & 9** .

4.1 Tour Definition

4.1.1 8.0, 9.0

Example

```
{
    id: 'mails_count_tour',
    name: _t("Mails count Tour"),
    mode: 'test',
    path: '/web#id=3&model=res.partner',
    steps: [
        {
            title:     _t("Mails count tutorial"),
            content:   _t("Let's see how mails count work."),
            popover:   { next: _t("Start Tutorial"), end: _t("Skip") },
        },
        {
            title:     _t("New fields"),
            content:   _t("Here is new fields with mails counters. Press one of it."),
            element:   '.mails_to',
        },
        {
            waitNot:   '.mails_to:visible',
            title:     _t("Send message from here"),
            placement: 'left',
            content:   _t("Now you can see corresponding mails. You can send mail to␣
↪this partner right from here. Press <em>'Send a mesage'</em>."),
            element:   '.oe_mail_wall .oe_msg.oe_msg_composer_compact>div>.oe_compose_
↪post',
```

(continues on next page)

(continued from previous page)

```
        },
    ]
}
```

Tour.register

In odoo 8 tour defines this way:

```
(function () {
'use strict';
var _t = openerp._t;
openerp.Tour.register({ ...
```

In odoo 9 tour defines that way:

```
odoo.define('account.tour_bank_statement_reconciliation', function(require) {
'use strict';
var core = require('web.core');
var Tour = require('web.Tour');
var _t = core._t;
Tour.register({ ...
```

Important details:

- **id** - need to call this tour
- **path** - from this path tour will be started in test mode

Step

Next step occurs when **all** conditions are satisfied and popup window will appear near (chose position in *placement*) element specified in *element*. Element must contain css selector of corresponding node. Conditions may be:

- **waitFor** - this step will not start if *waitFor* node absent.
- **waitNot** - this step will not start if *waitNot* node exists.
- **wait** - just wait some amount of milliseconds before **next** step.
- **element** - similar to *waitFor*, but *element* must be visible
- **closed window** - if popup window have close button it must be closed before next step.

Opened popup window (from previous step) will close automatically and new window (next step) will be shown.

Inject JS Tour file on page:

```
<template id="res_partner_mails_count_assets_backend" name="res_partner_mails_count_
↪assets_backend" inherit_id="web.assets_backend">
    <xpath expr="." position="inside">
        <script src="/res_partner_mails_count/static/src/js/res_partner_mails_count_
↪tour.js" type="text/javascript"></script>
    </xpath>
</template>
```

More documentation

Some docs is here (begin from 10 slide): http://www.slideshare.net/openobject/how-to-develop-automated-tests Also checkout here: https://github.com/odoo/odoo/blob/9.0/addons/web/static/src/js/tour.js

4.2 Open backend menu

4.2.1 8.0

The only way to open menu is search by string, for example

```
{
    title:      "go to accounting",
    element:    '.oe_menu_toggler:contains("Accounting"):visible',
},
```

4.2.2 9.0

Some additional actions are required to work with backend menus in tours

Manifest

Add `web_tour` to dependencies

```
"depends": [
    "web_tour",
],
# ...
"demo": [
    "views/assets_demo.xml",
    "views/tour_views.xml",
],
```

load_xmlid

You need to set `load_xmlid` for *each* menu you need to open. Recommended name for the file is `tour_views.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Make the xmlid of menus required by the tour available in webclient -->
    <record id="base.menu_administration" model="ir.ui.menu">
        <field name="load_xmlid" eval="True"/>
    </record>
</odoo>
```

Tour

Use *trigger* selector for both editions:

```
{
    trigger: '.o_app[data-menu-xmlid="base.menu_administration"], .oe_menu_
↪toggler[data-menu-xmlid="base.menu_administration"]',
    content: _t("Configuration options are available in the Settings app."),
    position: "bottom"
}
```

4.3 Manual Launching

4.3.1 8.0, 9.0

You can launch tour by url of following format:

`/web#/tutorial.mails_count_tour=true`

where *mails_count_tour* is id of your tour.

4.4 Auto Launch after installation

4.4.1 8.0, 9.0

To run tour after module installation do next steps.

- Create *ToDo*
- Create *Action*

ToDo is some queued web actions that may call *Action* like this:

```
<record id="base.open_menu" model="ir.actions.todo">
    <field name="action_id" ref="action_website_tutorial"/>
    <field name="state">open</field>
</record>
```

Action is like this:

```
<record id="res_partner_mails_count_tutorial" model="ir.actions.act_url">
    <field name="name">res_partner_mails_count Tutorial</field>
    <field name="url">/web#id=3&amp;model=res.partner&amp;/#tutorial_extra.mails_
↪count_tour=true</field>
    <field name="target">self</field>
</record>
```

Here tutorial_extra.**mails_count_tour** is tour id.

Use eval to compute some python code if needed:

```
<field name="url" eval="'/web?debug=1&amp;res_partner_mails_count=tutorial#id=
↪'+str(ref('base.partner_root'))+'&amp;view_type=form&amp;model=res.partner&amp;/
↪#tutorial_extra.mails_count_tour=true'"/>
```

Note: The later versions and all updates are available *here*.