
Odooly Documentation

Release 2.1.9

Florent Xicluna

Oct 20, 2021

Contents

1	Introduction	3
1.1	Installation	3
1.2	Command line arguments	3
1.3	Interactive use	4
2	Odooly API	7
2.1	Client and Services	7
2.2	Environment	9
2.3	Model and Records	11
2.4	Utilities	15
3	Tutorial	17
3.1	First connection	17
3.2	Create a database	18
3.3	Clone a database	19
3.4	Find the users	19
3.5	Create a new user	21
3.6	Explore the model	21
3.7	Browse the records	23
4	Developer's notes	25
4.1	Source code	25
4.2	Third-party integration	25
4.3	Changes	25
5	Indices and tables	29
6	Credits	31
	Python Module Index	33
	Index	35

A versatile tool for browsing Odoo / OpenERP data

The Odooly library communicates with any Odoo / OpenERP server (≥ 6.1) using the standard XML-RPC interface or the new JSON-RPC interface.

It provides both a *fully featured low-level API*, and an encapsulation of the methods on *Active Record objects*. It implements the Odoo API 8.0. Additional helpers are provided to explore the model and administrate the server remotely.

The *Introduction* describes how to use it as a *command line tool* or within an *interactive shell*.

The *Tutorial* gives an in-depth look at the capabilities.

Contents:

This section gives the bare minimum to use Odooly as a *command line tool* or within an *interactive shell*.

1.1 Installation

Download and install the [latest release](#) from PyPI:

```
pip install -U odooly
```

1.2 Command line arguments

There are few arguments to query Odoo models from the command line. Although it is quite limited:

```
$ odooly --help

Usage: odooly.py [options] [search_term_or_id [search_term_or_id ...]]

Inspect data on Odoo objects. Use interactively or query a model (-m) and
pass search terms or ids as positional parameters after the options.

Options:
  --version                show program's version number and exit
  -h, --help              show this help message and exit
  -l, --list              list sections of the configuration
  --env=ENV               read connection settings from the given section
  -c CONFIG, --config=CONFIG
                          specify alternate config file (default: 'odooly.ini')
  --server=SERVER         full URL of the server (default:
                          http://localhost:8069/xmlrpc)
  -d DB, --db=DB         database
```

(continues on next page)

(continued from previous page)

```
-u USER, --user=USER  username
-p PASSWORD, --password=PASSWORD
                        password, or it will be requested on login
-m MODEL, --model=MODEL
                        the type of object to find
-f FIELDS, --fields=FIELDS
                        restrict the output to certain fields (multiple
                        allowed)
-i, --interact         use interactively; default when no model is queried
-v, --verbose         verbose
$ #
```

Example:

```
$ odooly -d demo -m res.partner -f name -f lang 1
"name", "lang"
"Your Company", "en_US"
```

```
$ odooly -d demo -m res.groups -f full_name 'id > 0'
"full_name"
"Administration / Access Rights"
"Administration / Configuration"
"Human Resources / Employee"
"Usability / Multi Companies"
"Usability / Extended View"
"Usability / Technical Features"
"Sales Management / User"
"Sales Management / Manager"
"Partner Manager"
```

1.3 Interactive use

Edit `odooly.ini` and declare the environment(s):

```
[DEFAULT]
scheme = http
host = localhost
port = 8069
database = odoo
username = admin

[demo]
username = demo
password = demo
protocol = xmlrpc

[demo_jsonrpc]
username = demo
password = demo
protocol = jsonrpc

[local]
scheme = local
options = -c /path/to/odoo-server.conf --without-demo all
```


Connect to the Odoo server:

```
odooly --list
odooly --env demo
```

This is a sample session:

```
>>> env['res.users']
<Model 'res.users'>
>>> env['res.users'].search_count()
4
>>> crons = env['ir.cron'].with_context(active_test=False).search([])
>>> crons.read('active name')
[{'active': True, 'id': 5, 'name': 'Calendar: Event Reminder'},
 {'active': False, 'id': 4, 'name': 'Mail: Fetchmail Service'}]
>>> #
>>> env.modules('delivery')
{'uninstalled': ['delivery', 'website_sale_delivery']}
>>> env.upgrade('base')
1 module(s) selected
42 module(s) to process:
  to upgrade    account
  to upgrade    account_chart
  to upgrade    account_tax_include
  to upgrade    base
  ...
>>> #
```

Note: Use the `--verbose` switch to see what happens behind the scene. Lines are truncated at 79 chars. Use `-vv` or `-vvv` to print more.

Note: To preserve the history of commands when closing the session, first create an empty file in your home directory:
`touch ~/.odooly_history`

More details in the *Tutorial* section.

The library provides few objects to access the Odoo model and the associated services of the [Odoo API](#).

The signature of the methods mimics the standard methods provided by the `osv.Model` Odoo class. This is intended to help the developer when developing addons. What is experimented at the interactive prompt should be portable in the application with little effort.

- *Client and Services*
 - *Odoo RPC Services*
- *Environment*
 - *Advanced methods*
 - *Manage addons*
- *Model and Records*
- *Utilities*

2.1 Client and Services

The *Client* object provides thin wrappers around Odoo RPC services and their methods. Additional helpers are provided to explore the models and list or install Odoo add-ons.

```
class odooly.Client(server, db=None, user=None, password=None, transport=None, verbose=False)
    Connection to an Odoo instance.
```

This is the top level object. The *server* is the URL of the instance, like `http://localhost:8069`. If *server* is an `odoo/openerp` Python package, it is used to connect to the local server.

The *db* is the name of the database and the *user* should exist in the table `res.users`. If the *password* is not provided, it will be asked on login.

classmethod `Client.from_config(environment, user=None, verbose=False)`

Create a connection to a defined environment.

Read the settings from the section `[environment]` in the `odooly.ini` file and return a connected `Client`. See `read_config()` for details of the configuration file format.

`Client.create_database(password, database, demo=False, lang='en_US', user_password='admin', login='admin', country_code=None)`

Create a new database.

The superadmin `password` and the `database` name are mandatory. By default, `demo` data are not loaded, `lang` is `en_US` and no country is set into the database. Login if successful.

`Client.clone_database(password, database)`

Clone the current database.

The superadmin `password` and `database` are mandatory. Login if successful.

Supported since OpenERP 7.

`Client.login(user, password=None, database=None)`

Switch `user` and (optionally) `database`.

Note: In *interactive mode*, a method `Client.connect(env=None)` exists, to connect to another environment, and recreate the `globals()`.

Note: If the HTTPS server certificate is invalid, there's a trick to bypass the certificate verification, when the environment variable is set `ODOOLY_SSL_UNVERIFIED=1`.

2.1.1 Odoo RPC Services

The naked Odoo RPC services are exposed too. The `db` and the `common` services expose few methods which might be helpful for server administration. Use the `dir()` function to introspect them. The `_object` service should not be used directly because its methods are wrapped and exposed on the `Env` object itself. The two last services are deprecated and removed in recent versions of Odoo. Please refer to [the Odoo documentation](#) for more details.

`Client.db`

Expose the `db Service`.

Examples: `Client.db.list()` or `Client.db.server_version()` RPC methods.

`Client.common`

Expose the `common Service`.

Example: `Client.common.login_message()` RPC method.

`Client._object`

Expose the `object Service`.

`Client._report`

Expose the `report Service`.

Removed in Odoo 11.

`Client._wizard`

Expose the `wizard Service`.

Removed in OpenERP 7.

class `odooly.Service` (*client, endpoint, methods, verbose=False*)

A wrapper around XML-RPC endpoints.

The connected endpoints are exposed on the Client instance. The *server* argument is the URL of the server (scheme+host+port). If *server* is an `odoo` Python package, it is used to connect to the local server. The *endpoint* argument is the name of the service (examples: "object", "db"). The *methods* is the list of methods which should be exposed on this endpoint. Use `dir(...)` on the instance to list them.

2.2 Environment

class `odooly.Env`

An environment wraps data for Odoo models and records:

- *db_name*, the current database;
- *uid*, the current user id;
- *context*, the current context dictionary.

To retrieve an instance of `some.model`:

```
>>> env["some.model"]
```

db_name

Environment's database name.

uid

Environment's user id.

user

Instance of the environment's user.

context

Environment's context dictionary. It defaults to the `lang` and `tz` of the user. Use `Model.with_context()` to switch the context. For example `env['account.invoice'].with_context({})` can be used to call a method which does not accept the context argument.

cr

Cursor on the current database.

sudo (*user=SUPERUSER_ID*)

Attach to the provided user, or SUPERUSER.

__getitem__ (*name*)

Return the given `Model`.

access (*model_name, mode='read'*)

Check if the user has access to this model.

Optional argument *mode* is the access mode to check. Valid values are `read`, `write`, `create` and `unlink`. If omitted, the `read` mode is checked. Return a boolean.

execute (*obj, method, *params, **kwargs*)

Wrapper around `object.execute_kw` RPC method.

Argument *method* is the name of an `osv.osv` method or a method available on this *obj*. Method *params* are allowed. If needed, keyword arguments are collected in *kwargs*.

lang

Return the current language code.

models (*name=""*)

Search Odoo models.

The argument *name* is a pattern to filter the models returned. If omitted, all models are returned.

The return value is a sorted list of model names.

odoo_env

Return a server Environment.

Supported since Odoo 8.

ref (*xml_id*)

Return the record for the given `xml_id` external ID.

registry

Return the environment's registry.

Note: When connected to the local Odoo server, the `Env.odoo_env` attribute grabs an Odoo Environment with the same characteristics as the `Env` instance (`db_name`, `uid`, `context`). In this case a cursor on the database is available as `Env.cr`.

2.2.1 Advanced methods

Those methods give more control on the Odoo objects: workflows and reports. Please refer to [the Odoo documentation](#) for details.

`Env.execute` (*obj, method, *params, **kwargs*)

Wrapper around `object.execute_kw` RPC method.

Argument *method* is the name of an `osv.osv` method or a method available on this *obj*. Method *params* are allowed. If needed, keyword arguments are collected in *kwargs*.

`Env.exec_workflow` (*obj, signal, obj_id*)

Wrapper around `object.exec_workflow` RPC method.

Argument *obj* is the name of the model. The *signal* is sent to the object identified by its integer `id obj_id`.

Removed in Odoo 11.

`Env.report` (*obj, ids, datas=None*)

Wrapper around `report.report` RPC method.

Removed in Odoo 11.

`Env.render_report` (*obj, ids, datas=None*)

Wrapper around `report.render_report` RPC method.

Removed in Odoo 11.

`Env.report_get` (*report_id*)

Wrapper around `report.report_get` RPC method.

Removed in Odoo 11.

`Env.wizard_create` (*wiz_name, datas=None*)

Wrapper around `wizard.create` RPC method.

Removed in OpenERP 7.

Env.**wizard_execute** (*wiz_id, datas, action='init', context=None*)
 Wrapper around `wizard.execute` RPC method.
 Removed in OpenERP 7.

2.2.2 Manage addons

These helpers are convenient to list, install or upgrade addons from a Python script or interactively in a Python session.

Env.**modules** (*name=""*, *installed=None*)
 Return a dictionary of modules.

The optional argument *name* is a pattern to filter the modules. If the boolean argument *installed* is `True`, the modules which are “Not Installed” or “Not Installable” are omitted. If the argument is `False`, only these modules are returned. If argument *installed* is omitted, all modules are returned. The return value is a dictionary where module names are grouped in lists according to their *state*.

Env.**install** (**modules*)
 Press the button `Install`.

Env.**upgrade** (**modules*)
 Press the button `Upgrade`.

Env.**uninstall** (**modules*)
 Press the button `Uninstall`.

Note: It is not recommended to install or upgrade modules in offline mode when any web server is still running: the operation will not be signaled to other processes. This restriction does not apply when connected through XML-RPC or JSON-RPC.

2.3 Model and Records

The *Env* provides a high level API similar to the Odoo API, which encapsulates objects into *Active Records*.

The *Model* is instantiated using the `client.env[...]` syntax.

Example: `client.env['res.company']` returns a *Model*.

class `odooly.Model` (*client, model_name*)
 The class for Odoo models.

keys ()
 Return the keys of the model.

fields (*names=None, attributes=None*)
 Return a dictionary of the fields of the model.

Optional argument *names* is a sequence of field names or a space separated string of these names. If omitted, all fields are returned. Optional argument *attributes* is a sequence of attributes or a space separated string of these attributes. If omitted, all attributes are returned.

field (*name*)
 Return the field properties for field *name*.

access (*mode*='read')

Check if the user has access to this model.

Optional argument *mode* is the access mode to check. Valid values are read, write, create and unlink. If omitted, the read mode is checked. Return a boolean.

search (*domain*, *offset*=0, *limit*=None, *order*=None)

Search for records in the *domain*.

search_count (*domain*)

Count the records in the *domain*.

read (*domain*, *fields*=None, *offset*=0, *limit*=None, *order*=None)

Wrapper for `client.execute(model, 'read', [...], ('a', 'b'))`.

The first argument is a list of ids [1, 3] or a single id 42.

The second argument, *fields*, accepts:

- a single field: 'first_name'
- a tuple of fields: ('street', 'city')
- a space separated string: 'street city'
- a format spec: '%(street)s %(city)s'

If *fields* is omitted, all fields are read.

If *domain* is a single id, then:

- return a single value if a single field is requested.
- return a string if a format spec is passed in the *fields* argument.
- else, return a dictionary.

If *domain* is not a single id, the returned value is a list of items. Each item complies with the rules of the previous paragraph.

The optional keyword arguments *offset*, *limit* and *order* are used to restrict the search. The *order* is also used to order the results returned. Note: the low-level RPC method `read` itself does not preserve the order of the results.

get (*domain*)

Return a single *Record*.

The argument *domain* accepts a single integer *id* or a search domain, or an external ID `xml_id`. The return value is a *Record* or None. If multiple records are found, a `ValueError` is raised.

browse (*ids*)

Return a *Record* or a *RecordList*.

The argument *ids* accepts a single integer *id* or a list of ids. If it is a single integer, the return value is a *Record*. Otherwise, the return value is a *RecordList*.

create (*values*)

Create a *Record*.

The argument *values* is a dictionary of values which are used to create the record. Relationship fields *one2many* and *many2many* accept either a list of ids or a *RecordList* or the extended Odoo syntax. Relationship fields *many2one* and *reference* accept either a *Record* or the Odoo syntax.

The newly created *Record* is returned.

with_env (*env*)
 Attach to the provided environment.

sudo (*user=SUPERUSER_ID*)
 Attach to the provided user, or SUPERUSER.

with_context (*[context]*, ***overrides*)
 Attach to an extended context.

_get_external_ids (*ids=None*)
 Retrieve the External IDs of the records.

Return a dictionary with keys being the fully qualified External IDs, and values the `Record` entries.

class `odooly.RecordList` (*model, ids*)
 A sequence of Odoo `Record`.

It has a similar API as the `Record` class, but for a list of records. The attributes of the `RecordList` are read-only, and they return list of attribute values in the same order. The `many2one`, `one2many` and `many2many` attributes are wrapped in `RecordList` and list of `RecordList` objects. Use the method `RecordList.write` to assign a single value to all the selected records.

read (*fields=None*)
 Wrapper for the `Record.read()` method.

Return a `RecordList` if *fields* is the name of a single `many2one` field, else return a list. See `Model.read()` for details.

write (*values*)
 Wrapper for the `Record.write()` method.

unlink ()
 Wrapper for the `Record.unlink()` method.

exists ()
 Return a subset of records that exist.

mapped (*func*)
 Apply *func* on all records.

filtered (*func*)
 Select the records such that `func(rec)` is true.

As an alternative *func* can be a search domain (list) to search among the records.

sorted (*key=None, reverse=False*)
 Return the records sorted by *key*.

ensure_one ()
 Return the single record in this recordset.

Raise a `ValueError` if recordset has more records or is empty.

union (**args*)
 Return the union of all records.

Preserve first occurrence order.

concat (**args*)
 Return the concatenation of all records.

with_env (*env*)
 Attach to the provided environment.

sudo (*user=SUPERUSER_ID*)

Attach to the provided user, or SUPERUSER.

with_context (*[context]*, ***overrides*)

Attach to an extended context.

get_metadata ()

Wrapper for the *Record.get_metadata()* method.

_external_id

Retrieve the External IDs of the *RecordList*.

Return the list of fully qualified External IDs of the *RecordList*, with default value False if there's none. If multiple IDs exist for a record, only one of them is returned.

class *odooly.Record* (*model, id*)

A class for all Odoo records.

It maps any Odoo object. The fields can be accessed through attributes. The changes are immediately sent to the server. The *many2one*, *one2many* and *many2many* attributes are wrapped in *Record* and *RecordList* objects. These attributes support writing too. The attributes are evaluated lazily, and they are cached in the record. The *Record*'s cache is invalidated if any attribute is changed.

exists ()

Return a subset of records that exist.

get_metadata (*details=True*)

Lookup metadata about the record(s). Return dictionaries with the following keys:

- *id*: object id
- *create_uid*: user who created the record
- *create_date*: date when the record was created
- *write_uid*: last user who changed the record
- *write_date*: date of the last change to the record
- *xmlid*: External ID to use to refer to this record (if there is one), in format *module.name*.

_external_id

Retrieve the External ID of the *Record*.

Return the fully qualified External ID of the *Record*, with default value False if there's none. If multiple IDs exist, only one of them is returned (randomly).

_send (*signal*)

Trigger workflow *signal* for this *Record*.

copy (*default=None*)

Copy a record and return the new *Record*.

The optional argument *default* is a mapping which overrides some values of the new record.

read (*fields=None*)

Read the *fields* of the *Record*.

The argument *fields* accepts different kinds of values. See *Model.read()* for details.

refresh ()

Force refreshing the record's data.

unlink ()

Delete the record(s) from the database.

write (*values*)

Write the *values* in the record(s).

values is a dictionary of values. See `Model.create()` for details.

2.4 Utilities

`odooly.issearchdomain` (*arg*)

Check if the argument is a search domain.

Examples:

- `[('name', '=', 'mushroom'), ('state', '!=', 'draft')]`
- `['name = mushroom', 'state != draft']`
- `[]`

`odooly.searchargs` (*params, kwargs=None*)

Compute the 'search' parameters.

`odooly.format_exception` (*type, value, tb, limit=None, chain=True*)

Format a stack trace and the exception information.

This wrapper is a replacement of `traceback.format_exception` which formats the error and traceback received by XML-RPC/JSON-RPC. If *chain* is `True`, then the original exception is printed too.

`odooly.read_config` (*section=None*)

Read the environment settings from the configuration file.

The config file `odooly.ini` contains a *section* for each environment. Each section provides parameters for the connection: `host`, `port`, `database`, `username` and (optional) `password`. Default values are read from the `[DEFAULT]` section. If the `password` is not in the configuration file, it is requested on login. Return a tuple (`server`, `db`, `user`, `password` or `None`). Without argument, it returns the list of configured environments.

`odooly.start_odoo_services` (*options=None, appname=None*)

Initialize the Odoo services.

Import the `odoo` Python package and load the Odoo services. The argument *options* receives the command line arguments for `odoo`. Example:

```
['-c', '/path/to/odoo-server.conf', '--without-demo', 'all'].
```

Return the `odoo` package.

This tutorial demonstrates some features of Odooly in the interactive shell.

It assumes an Odoo or OpenERP server is installed. The shell is a true Python shell. We have access to all the features and modules of the Python interpreter.

Steps:

- *First connection*
- *Create a database*
- *Clone a database*
- *Find the users*
- *Create a new user*
- *Explore the model*
- *Browse the records*

3.1 First connection

The server is freshly installed and does not have an Odoo database yet. The tutorial creates its own database demo to play with.

Open the Odooly shell:

```
$ odooly
```

It assumes that the server is running locally, and listens on default port 8069.

If our configuration is different, then we use arguments, like:

```
$ odooly --server http://192.168.0.42:8069
```

It connects using the XML-RPC protocol. If you want to use the JSON-RPC protocol instead, then pass the full URL with `/jsonrpc` path:

```
$ odooly --server http://127.0.0.1:8069/jsonrpc
```

On login, it prints few lines about the commands available.

```
$ odooly
Usage (some commands):
  env[name]                                # Return a Model instance
  env[name].keys()                          # List field names of the model
  env[name].fields(names=None)              # Return details for the fields
  env[name].field(name)                     # Return details for the field
  env[name].browse(ids=())                  # Return a RecordList
  env[name].search(domain)
  env[name].search(domain, offset=0, limit=None, order=None)

  rec = env[name].get(domain)                # Get the Record matching domain
  rec.some_field                            # Return the value of this field
  rec.read(fields=None)                     # Return values for the fields

  client.login(user)                        # Login with another user
  client.connect(env)                       # Connect to another env.
  env.models(name)                          # List models matching pattern
  env.modules(name)                         # List modules matching pattern
  env.install(module1, module2, ...)
  env.upgrade(module1, module2, ...)
  # Install or upgrade the modules
```

And it confirms that the default database is not available:

```
...
Error: Database 'odoo' does not exist: []
```

Though, we have a connected client, ready to use:

```
>>> client
<Client 'http://localhost:8069/xmlrpc#()'>
>>> client.server_version
'6.1'
>>> #
```

3.2 Create a database

We create the database "demo" for this tutorial. We need to know the superadmin password before to continue. This is the `admin_passwd` in the `odoo-server.conf` file. Default password is "admin".

Note: This password gives full control on the databases. Set a strong password in the configuration to prevent unauthorized access.

```

>>> client.create_database('super_password', 'demo')
Logged in as 'admin'
>>> client
<Client 'http://localhost:8069/xmlrpc#demo'>
>>> client.db.list()
['demo']
>>> env
<Env 'admin@demo'>
>>> env.modules(installed=True)
{'installed': ['base', 'web', 'web_mobile', 'web_tests']}
>>> len(env.modules()['uninstalled'])
202
>>> #

```

Note: Create an `odooly.ini` file in the current directory to declare all our environments. Example:

```

[DEFAULT]
host = localhost
port = 8069

[demo]
database = demo
username = joe

```

Then we connect to any environment with `odooly --env demo` or switch during an interactive session with `client.connect('demo')`.

3.3 Clone a database

It is sometimes useful to clone a database (testing, backup, migration, ...). A shortcut is available for that, the required parameters are the new database name and the superadmin password.

```

>>> client.clone_database('super_password', 'demo_test')
Logged in as 'admin'
>>> client
<Client 'http://localhost:8069/xmlrpc#demo_test'>
>>> client.db.list()
['demo', 'demo_test']
>>> env
<Env 'admin@demo'>
>>> client.modules(installed=True)
{'installed': ['base', 'web', 'web_mobile', 'web_tests']}
>>> len(client.modules()['uninstalled'])
202
>>> #

```

3.4 Find the users

We have created the database "demo" for the tests. We are connected to this database as 'admin'.

Where is the table for the users?

```
>>> client
<Client 'http://localhost:8069/xmlrpc#demo'>
>>> env.models('user')
['res.users', 'res.users.log']
```

We've listed two models which matches the name, `res.users` and `res.users.log`. Through the environment `Env` we reach the users' model and we want to introspect its fields. Fortunately, the `Model` class provides methods to retrieve all the details.

```
>>> env['res.users']
<Model 'res.users'>
>>> print(env['res.users'].keys())
['action_id', 'active', 'company_id', 'company_ids', 'context_lang',
 'context_tz', 'date', 'groups_id', 'id', 'login', 'menu_id', 'menu_tips',
 'name', 'new_password', 'password', 'signature', 'user_email', 'view']
>>> env['res.users'].field('company')
{'change_default': False,
 'company_dependent': False,
 'context': {'user_preference': True},
 'depends': [],
 'domain': [],
 'help': 'The company this user is currently working for.',
 'manual': False,
 'readonly': False,
 'relation': 'res.company',
 'required': True,
 'searchable': True,
 'sortable': True,
 'store': True,
 'string': 'Company',
 'type': 'many2one'}
>>> #
```

Let's examine the 'admin' user in details.

```
>>> env['res.users'].search_count()
1
>>> admin_user = env['res.users'].browse(1)
>>> admin_user.groups_id
<RecordList 'res.groups,[1, 2, 3]'>
>>> admin_user.groups_id.name
['Access Rights', 'Configuration', 'Employee']
>>> admin_user.groups_id.full_name
['Administration / Access Rights',
 'Administration / Configuration',
 'Human Resources / Employee']
>>> admin_user.get_metadata()
{'create_date': False,
 'create_uid': False,
 'id': 1,
 'write_date': '2012-09-01 09:01:36.631090',
 'write_uid': [1, 'Administrator'],
 'xmlid': 'base.user_admin'}
```


3.5 Create a new user

Now we want a non-admin user to continue the exploration. Let's create Joe.

```
>>> env['res.users'].create({'login': 'joe'})
Fault: Integrity Error

The operation cannot be completed, probably due to the following:
- deletion: you may be trying to delete a record while other records still reference_
↳it
- creation/update: a mandatory field is not correctly set

[object with reference: name - name]
>>> #
```

It seems we've forgotten some mandatory data. Let's give him a name.

```
>>> env['res.users'].create({'login': 'joe', 'name': 'Joe'})
<Record 'res.users,3'>
>>> joe_user = _
>>> joe_user.groups_id.full_name
['Human Resources / Employee', 'Partner Manager']
```

The user Joe does not have a password: we cannot login as joe. We set a password for Joe and we try again.

```
>>> client.login('joe')
Password for 'joe':
Error: Invalid username or password
>>> env.user
'admin'
>>> joe_user.password = 'bar'
>>> client.login('joe')
Logged in as 'joe'
>>> env.user
'joe'
>>> #
```

Success!

3.6 Explore the model

We keep connected as user Joe and we explore the world around us.

```
>>> env.user
'joe'
>>> all_models = env.models()
>>> len(all_models)
92
```

Among these 92 objects, some of them are read-only, others are read-write. We can also filter the non-empty models.

```
>>> # Read-only models
>>> len([m for m in all_models if not env[m].access('write')])
44
```

(continues on next page)

```

>>> #
>>> # Writable but cannot delete
>>> [m for m in all_models if env[m].access('write') and not env[m].access('unlink')]
['ir.property', 'web.planner']
>>> #
>>> # Unreadable models
>>> [m for m in all_models if not env[m].access('read')]
['ir.actions.todo',
 'ir.actions.todo.category',
 'res.payterm']
>>> #
>>> # Now print the number of entries in all (readable) models
>>> for m in all_models:
...     mcount = env[m].access() and env[m].search_count()
...     if not mcount:
...         continue
...     print('%4d %s' % (mcount, m))
...
   1  ir.actions.act_url
  64  ir.actions.act_window
  14  ir.actions.act_window.view
  76  ir.actions.act_window_close
  76  ir.actions.actions
   4  ir.actions.client
   4  ir.actions.report
   3  ir.actions.server
   1  ir.default
 112  ir.model
3649  ir.model.data
1382  ir.model.fields
   33  ir.ui.menu
  221  ir.ui.view
   3  report.paperformat
   1  res.company
  249  res.country
   2  res.country.group
  678  res.country.state
   2  res.currency
   9  res.groups
   1  res.lang
   5  res.partner
  21  res.partner.industry
   5  res.partner.title
   1  res.request.link
   4  res.users
  12  res.users.log
>>> #
>>> # Show the content of a model
>>> config_params = env['ir.config_parameter'].search([])
>>> config_params.read('key value')
[{'id': 1, 'key': 'web.base.url', 'value': 'http://localhost:8069'},
 {'id': 2, 'key': 'database.create_date', 'value': '2012-09-01 09:01:12'},
 {'id': 3,
  'key': 'database.uuid',
  'value': '52fc9630-f49e-2222-e622-08002763afeb'}]

```

3.7 Browse the records

Query the "res.country" model:

```
>>> env['res.country'].keys()
['address_format', 'code', 'name']
>>> env['res.country'].search(['name like public'])
<RecordList 'res.country',[41, 42, 57, 62, 144]>
>>> env['res.country'].search(['name like public']).name
['Central African Republic',
 'Congo, Democratic Republic of the',
 'Czech Republic',
 'Dominican Republic',
 'Macedonia, the former Yugoslav Republic of']
>>> env['res.country'].search(['code > Y'], order='code ASC').read('code name')
[{'code': 'YE', 'id': 247, 'name': 'Yemen'},
 {'code': 'YT', 'id': 248, 'name': 'Mayotte'},
 {'code': 'ZA', 'id': 250, 'name': 'South Africa'},
 {'code': 'ZM', 'id': 251, 'name': 'Zambia'},
 {'code': 'ZW', 'id': 253, 'name': 'Zimbabwe'}]
>>> #
```

... the tutorial is done.

Jump to the *Odooly API* for further details.

4.1 Source code

- Source code and issue tracker on [GitHub](#).
- Continuous tests against Python 2.7, 3.5 through 3.9 and PyPy, on [Travis-CI](#) platform.

4.2 Third-party integration

This module can be used with other Python libraries to achieve more complex tasks.

For example:

- write unit tests using the standard [unittest](#) framework.
- write BDD tests using the [Gherkin](#) language, and a library like [Behave](#).
- build an interface for [Odoo](#), using a framework like [Flask](#) (HTML, JSON, SOAP, ...).

4.3 Changes

4.3.1 2.x.x (unreleased)

- Drop support for Python 3.4

4.3.2 2.1.9 (2019-10-02)

- No change. Re-upload to PyPI.

4.3.3 2.1.8 (2019-10-02)

- Default location for the configuration file is the initial working directory.
- Enhanced syntax for method `RecordList.filtered()`. E.g. instead of `records.filtered(lambda r: r.type == 'active')` it's faster to use `records.filtered(['type = active'])`.
- Support unary operators even for Python 3.
- Basic sequence operations on `Env` instance.

4.3.4 2.1.7 (2019-03-20)

- No change. Re-upload to PyPI.

4.3.5 2.1.6 (2019-03-20)

- Fix `RecordList.mapped()` method with empty one2many or many2many fields.
- Hide arguments of partial objects.

4.3.6 2.1.5 (2019-02-12)

- Fix new feature of 2.1.4.

4.3.7 2.1.4 (2019-02-12)

- Support `env['res.partner'].browse()` and return an empty `RecordList`.

4.3.8 2.1.3 (2019-01-09)

- Fix a bug where method `with_context` returns an error if we update the values of the logged-in user before.
- Allow to call RPC method `env['ir.default'].get(...)` thanks to a passthrough in the `Model.get()` method.

4.3.9 2.1.2 (2019-01-02)

- Store the cursor `Env.cr` on the `Env` instance in local mode.
- Drop support for Python 3.2 and 3.3

4.3.10 2.1.1 (2019-01-02)

- Do not call ORM method `exists` on an empty list because it fails with OpenERP.
- Provide cursor `Env.cr` in local mode, even with OpenERP instances.
- Optimize and fix method `RecordList.filtered()`.

4.3.11 2.1 (2018-12-27)

- Allow to bypass SSL verification if the server is misconfigured. Environment variable `ODOOLY_SSL_UNVERIFIED=1` is detected.
- Accept multiple command line arguments for local mode. Example: `odooly -- --config path/to/odoo.conf --data-dir ./var`
- Add `self` to the `globals()` in interactive mode, to mimic Odoo shell.
- On login, assign the context of the user: `env['res.users'].context_get()`. Do not copy the context when switching database, or when connecting with a different user.
- Drop attribute `Client.context`. It is only available as `Env.context`.
- Fix hashing error when `Env.context` contains a list.
- Assign the model name to `Record._name`.
- Fix installation/upgrade with an empty list.
- Catch error when database does not exist on login.
- Format other Odoo errors like `DatabaseExists`.

4.3.12 2.0 (2018-12-12)

- Fix cache of first `Env` in interactive mode.
- Correctly invalidate the cache after installing/upgrading add-ons.
- Add tests for `Model.with_context()`, `Model.sudo()` and `Env.sudo()`.
- Copy the context when switching database.
- Change interactive prompt `sys.ps2` to " ... ".

4.3.13 2.0b3 (2018-12-10)

- Provide `Env.sudo()` in addition to same method on `Model`, `RecordList` and `Record` instances.
- Workflows and method `object.exec_workflow` are removed in Odoo 11.
- Do not prevent login if access to `Client.db.list()` is denied.
- Use a cache of `Env` instances.

4.3.14 2.0b2 (2018-12-05)

- Add documentation for methods `RecordList.exists()` and `RecordList.ensure_one()`.
- Add documentation for methods `RecordList.mapped()`, `RecordList.filtered()` and `RecordList.sorted()`.
- Add documentation for methods `Model.with_env()`, `Model.sudo()` and `Model.with_context()`. These methods are also available on `RecordList` and `Record`.
- Changed method `exists` on `RecordList` and `Record` to return record(s) instead of ids.
- Fix methods `mapped`, `filtered` and `sorted`. Add tests.
- Fix method `RecordList.ensure_one()` when there's identical ids or `False` values.

- Fix method `RecordList.union(...)` and related boolean operations.

4.3.15 2.0b1 (2018-12-04)

- First release of Odooly, which mimics the new Odoo 8.0 API.
- Other features are copied from [ERPpeek 1.7](#).
- Online documentation: <http://odooly.readthedocs.org/>
- Source code and issue tracker: <https://github.com/tinyerp/odooly>

CHAPTER 5

Indices and tables

- `genindex`
- `search`

CHAPTER 6

Credits

Authored and maintained by Florent Xicluna.

O

odooly, 7

Symbols

__getitem__() (*odooly.Env method*), 9
 _external_id (*odooly.Record attribute*), 14
 _external_id (*odooly.RecordList attribute*), 14
 _get_external_ids() (*odooly.Model method*), 13
 _report (*odooly.Client attribute*), 8
 _send() (*odooly.Record method*), 14
 _wizard (*odooly.Client attribute*), 8

A

access() (*odooly.Env method*), 9
 access() (*odooly.Model method*), 11

B

browse() (*odooly.Model method*), 12

C

Client (*class in odooly*), 7
 Client._object (*in module odooly*), 8
 clone_database() (*odooly.Client method*), 8
 common (*odooly.Client attribute*), 8
 concat() (*odooly.RecordList method*), 13
 context (*odooly.Env attribute*), 9
 copy() (*odooly.Record method*), 14
 cr (*odooly.Env attribute*), 9
 create() (*odooly.Model method*), 12
 create_database() (*odooly.Client method*), 8

D

db (*odooly.Client attribute*), 8
 db_name (*odooly.Env attribute*), 9

E

ensure_one() (*odooly.RecordList method*), 13
 Env (*class in odooly*), 9
 exec_workflow() (*odooly.Env method*), 10
 execute() (*odooly.Env method*), 9, 10
 exists() (*odooly.Record method*), 14
 exists() (*odooly.RecordList method*), 13

F

field() (*odooly.Model method*), 11
 fields() (*odooly.Model method*), 11
 filtered() (*odooly.RecordList method*), 13
 format_exception() (*in module odooly*), 15
 from_config() (*odooly.Client class method*), 7

G

get() (*odooly.Model method*), 12
 get_metadata() (*odooly.Record method*), 14
 get_metadata() (*odooly.RecordList method*), 14

I

install() (*odooly.Env method*), 11
 issearchdomain() (*in module odooly*), 15

K

keys() (*odooly.Model method*), 11

L

lang (*odooly.Env attribute*), 9
 login() (*odooly.Client method*), 8

M

mapped() (*odooly.RecordList method*), 13
 Model (*class in odooly*), 11
 models() (*odooly.Env method*), 9
 modules() (*odooly.Env method*), 11

O

odoo_env (*odooly.Env attribute*), 10
 odooly (*module*), 7

R

read() (*odooly.Model method*), 12
 read() (*odooly.Record method*), 14
 read() (*odooly.RecordList method*), 13
 read_config() (*in module odooly*), 15

Record (*class in odooly*), 14
RecordList (*class in odooly*), 13
ref () (*odooly.Env method*), 10
refresh () (*odooly.Record method*), 14
registry (*odooly.Env attribute*), 10
render_report () (*odooly.Env method*), 10
report () (*odooly.Env method*), 10
report_get () (*odooly.Env method*), 10

S

search () (*odooly.Model method*), 12
search_count () (*odooly.Model method*), 12
searchargs () (*in module odooly*), 15
Service (*class in odooly*), 8
sorted () (*odooly.RecordList method*), 13
start_odoo_services () (*in module odooly*), 15
sudo () (*odooly.Env method*), 9
sudo () (*odooly.Model method*), 13
sudo () (*odooly.RecordList method*), 13

U

uid (*odooly.Env attribute*), 9
uninstall () (*odooly.Env method*), 11
union () (*odooly.RecordList method*), 13
unlink () (*odooly.Record method*), 14
unlink () (*odooly.RecordList method*), 13
upgrade () (*odooly.Env method*), 11
user (*odooly.Env attribute*), 9

W

with_context () (*odooly.Model method*), 13
with_context () (*odooly.RecordList method*), 14
with_env () (*odooly.Model method*), 12
with_env () (*odooly.RecordList method*), 13
wizard_create () (*odooly.Env method*), 10
wizard_execute () (*odooly.Env method*), 10
write () (*odooly.Record method*), 14
write () (*odooly.RecordList method*), 13