# OERPLib Documentation

***Release 0.8***

**ABF Osiell**

March 04, 2014

CHAPTER 1

# Introduction

**OERPLib** is a *RPC* client library to **OpenERP** server written in Python. It aims to provide an easy way to remotely pilot an *OpenERP* server.

**Features supported:**

- *XML-RPC* and (legacy) *Net-RPC* protocols,
- access to all methods proposed by an *OpenERP* model class (even `browse`) with an API similar to the server-side API,
- ability to use named parameters with such methods (*OpenERP >= 6.1*),
- user context automatically sent (*OpenERP >= 6.1*) providing support for internationalization,
- browse records,
- execute workflows,
- manage databases,
- reports downloading,
- inspection capabilities (graphical output of relations between models and dependencies between modules, list `on_change` methods from model views, ...).

CHAPTER 2

# Quick start

How does it work? See below:

```
import oerplib

# Prepare the connection to the OpenERP server
oerp = oerplib.OERP('localhost', protocol='xmlrpc', port=8069)

# Check available databases
print(oerp.db.list())

# Login (the object returned is a browsable record)
user = oerp.login('user', 'passwd', 'db_name')
print(user.name)            # name of the user connected
print(user.company_id.name) # the name of its company

# Simple 'raw' query
user_data = oerp.execute('res.users', 'read', [user.id])
print(user_data)

# Use all methods of a model class
order_obj = oerp.get('sale.order')
order_ids = order_obj.search([])
for order in order_obj.browse(order_ids):
    print(order.name)
    products = [line.product_id.name for line in order.order_line]
    print(products)

# Update data through a browsable record
user.name = "Brian Jones"
oerp.write_record(user)
```

For more details and features, see the *tutorials*, the *Frequently Asked Questions (FAQ)* and the *API reference* sections.

CHAPTER 3

# Contents

## 3.1 Download and install instructions

### 3.1.1 Python Package Index (PyPI)

You can install OERPLib with the *easy_install* tool:

```
$ easy_install oerplib
```

Or with *pip*:

```
$ pip install oerplib
```

An alternative way is to download the tarball from Python Package Index page, and install manually (replace *X.Y.Z* accordingly):

```
$ wget http://pypi.python.org/packages/source/O/OERPLib/OERPLib-X.Y.Z.tar.gz
$ tar xzvf OERPLib-X.Y.Z.tar.gz
$ cd OERPLib-X.Y.Z
$ python setup.py install
```

No dependency is required except pydot for some methods of the `inspect` service (optional).

### 3.1.2 Source code

The project is hosted on Launchpad. To get the current development branch (the `trunk`), just type:

```
$ bzr branch lp:oerplib
```

For the last version of a stable branch (replace *X.Y* accordingly):

```
$ bzr branch lp:oerplib/X.Y
```

### 3.1.3 Run tests

New in version 0.4.0.

Unit tests depends on *unittest2* (Python 2.3+) or *unittest* (Python 2.7 and 3.x), and *argparse*.

To run unit tests from the project directory, run the following command:

```
$ PYTHONPATH=.:$PYTHONPATH ./tests/runtests.py --help
```

Then, set your parameters in order to indicate the *OpenERP* server on which you want to perform the tests, for instance:

```
$ PYTHONPATH=.:$PYTHONPATH ./tests/runtests.py --create_db --server 192.168.1.4 --test_xmlrpc --xmlrp
```

The name of the database created is `oerplib-test` by default.

## 3.2 Tutorials

### 3.2.1 First step: prepare the connection and login

You need an instance of the `OERP` class to dialog with an *OpenERP* server. Let's pretend that you want to connect as *admin* on the *db_name* database of the local *OpenERP* server (with the *XML-RPC* service which listens on port *8071*). First, prepare the connection:

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost', protocol='xmlrpc', port=8071)
```

You can also specify the default database to use with the *database* parameter:

```
>>> oerp = oerplib.OERP('localhost', database='db_name', protocol='xmlrpc', port=8071)
```

To check databases available, use the `oerp.db` attribute with the **list** method:

```
>>> oerp.db.list()
['db_name', 'db_name2', ...]
```

The connection is ready, you are able to log in on the server with the account of your choice:

```
>>> user = oerp.login(user='admin', passwd='admin')
```

Or, if no default database was specified before:

```
>>> user = oerp.login(user='admin', passwd='admin', database='db_name')
```

The `login` method returns an object representing the user connected. It is built from the server-side model `res.users`, and all its informations are accessible (see *Browse records* section):

```
>>> print(user.name)             # print the full name of the user
>>> print(user.company_id.name) # print the name of its company
```

Now you are connected, you can easily execute any kind of *RPC* queries on the the *OpenERP* server (execute model methods, trigger workflow, download reports, and handle wizards).

### 3.2.2 Execute queries

The basic method to execute queries (related to the `/object` *RPC* service) is `execute`. It takes at least two parameters (model name and the method name) following by variable parameters according to the method called. Example:

```
>>> order_data = oerp.execute('sale.order', 'read', [1], ['name'])
```

This instruction will call the `read` method of the model `sale.order` with the parameters `[1]` (list of record IDs) and `['name']` (list of fields to return).

However, for usual methods such as `create`, `read`, `write`, `unlink` and `search` there are convenient shortcuts available (see `oerplib.OERP`):

```
>>> partner_id = oerp.create('res.partner', {'name': 'Jacky Bob', 'lang': 'fr_FR'})
>>> partner_data = oerp.read('res.partner', [partner_id], ['name'])
>>> oerp.write('res.partner', [partner_id], {'name': 'Charly Bob'})
True
>>> partner_ids = oerp.search('res.partner', [('name', 'ilike', 'Bob')])
>>> oerp.unlink('res.partner', [partner_id])
True
```

There is another way to perform all methods of a model, with the `get` method, which provide an API almost syntactically identical to the *OpenERP* server side API (see `oerplib.service.osv.Model`):

```
>>> user_obj = oerp.get('res.users')
>>> user_obj.write([1], {'name': "Dupont D."})
True
>>> context = user_obj.context_get()
>>> context
{'lang': 'fr_FR', 'tz': False}
>>> product_obj = oerp.get('product.product')
>>> product_obj.name_get([3, 4])
[[3, '[PC1] PC Basic'], [4, u'[PC2] Basic+ PC (assembl\xe9 sur commande)']]
```

If you run an *OpenERP* version *6.1* or above, the user context is automatically sent. You can disable this behaviour with the `oerplib.OERP.config` property:

```
>>> oerp.config['auto_context'] = False
>>> product_obj.name_get([3, 4])    # Without context, lang 'en_US' by default
[[3, '[PC1] Basic PC'], [4, '[PC2] Basic+ PC (assembly on order)']]
```

**Note:** The `auto_context` option only affect model methods.

Here is another example of how to install a module (you have to be logged as an administrator to perform this task). The `button_immediate_install` method used here is available since *OpenERP v6.1*:

```
>>> module_obj = oerp.get('ir.module.module')
>>> module_id = module_obj.search([('name', '=', 'purchase')])
>>> module_obj.button_immediate_install(module_id)
```

### 3.2.3 Browse records

A great functionality of *OERPLib* is its ability to generate objects that are similar to browsable records found on the *OpenERP* server. All this is possible using the `browse` method:

```
# fetch one record
partner = oerp.browse('res.partner', 1) # Partner ID = 1
print(partner.name)
# fetch several records
for partner in oerp.browse('res.partner', [1, 2]):
    print(partner.name)
```

From such objects, it is possible to easily explore relationships. The related records are generated on the fly:

```
partner = oerp.browse('res.partner', 3)
for child in partner.child_ids:
    print(child.name)
```

You can browse objects through a `model` too. In fact, both methods are strictly identical, `oerplib.OERP.browse()` is simply a shortcut to the other:

```
>>> partner1 = oerp.browse('res.partner', 3)
>>> partner2 = oerp.get('res.partner').browse(3)
>>> partner1 == partner2
True
```

Outside relation fields, Python data types are used, like `datetime.date` and `datetime.datetime`:

```
>>> order = oerp.browse('purchase.order', 42)
>>> order.minimum_planned_date
datetime.datetime(2012, 3, 10, 0, 0)
>>> order.date_order
datetime.date(2012, 3, 8)
```

A list of data types used by `browse_record` fields are available *here*.

### 3.2.4 Update data through browsable records

Update data of a browsable record is workable with the `write_record` method of an `OERP` instance. Let's update the name of a partner:

```
>>> partner.name = "Caporal Jones"
>>> oerp.write_record(partner)
```

This is equivalent to:

```
>>> oerp.write('res.partner', [partner.id], {'name': "Caporal Jones"})
```

#### Char, Float, Integer, Boolean, Text and Binary

As see above, it's as simple as that:

```
>>> partner.name = "OpenERP"
>>> oerp.write_record(partner)
```

#### Selection

Same as above, except there is a check about the value assigned. For instance, the field `type` of the `res.partner` model accept values contains in `['default', 'invoice', 'delivery', 'contact', 'other']`:

```
>>> partner.type = 'default' # Ok
>>> partner.type = 'foobar'  # Error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "oerplib/fields.py", line 58, in setter
    value = self.check_value(value)
  File "oerplib/fields.py", line 73, in check_value
    field_name=self.name,
ValueError: The value 'foobar' supplied doesn't match with the possible values '['default', 'invoice'
```

#### Many2One

You can also update a `many2one` field, with either an ID or a browsable record:

```
>>> partner.parent_id = 1 # with an ID
>>> oerp.write_record(partner)
>>> parent = oerp.browse('res.partner', 1) # with a browsable record
>>> partner.parent_id = parent
>>> oerp.write_record(partner)
```

You can't put any ID or browsable record, a check is made on the relationship to ensure data integrity:

```
>>> user = oerp.browse('res.users', 1)
>>> partner = oerp.browse('res.partner', 2)
>>> partner.parent_id = user
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "oerplib/fields.py", line 128, in setter
    o_rel = self.check_value(o_rel)
  File "oerplib/fields.py", line 144, in check_value
    field_name=self.name))
ValueError: Instance of 'res.users' supplied doesn't match with the relation 'res.partner' of the 'pa
```

### One2Many and Many2Many

`one2many` and `many2many` fields can be updated by providing a list of tuple as specified in the *OpenERP* documentation, a list of records, a list of record IDs or an empty list or `False`:

With a standard *OpenERP* tuple, no magic here:

```
>>> user = oerp.get('res.users').browse(1)
>>> user.groups_id = [(6, 0, [8, 5, 6, 4])]
>>> oerp.write_record(user)
```

With a list of records:

```
>>> user = oerp.get('res.users').browse(1)
>>> groups = oerp.get('res.groups').browse([8, 5, 6, 4])
>>> user.groups_id = list(groups)
>>> oerp.write_record(user)
```

With a list of record IDs:

```
>>> user = oerp.get('res.users').browse(1)
>>> user.groups_id = [8, 5, 6, 4]
>>> oerp.write_record(user)
```

The last two examples are equivalent to the first (they generate a `(6, 0, IDS)` tuple).

However, if you set an empty list or `False`, a `(5, )` tuple will be generated to cut the relation between records:

```
>>> user = oerp.get('res.users').browse(1)
>>> user.groups_id = []
>>> list(user.groups_id)
[]
>>> user.__data__['updated_values']['groups_id']
[(5,)]
>>> user.groups_id = False
>>> list(user.groups_id)
[]
>>> user.__data__['updated_values']['groups_id']
[(5,)]
```

Another facility provided by *OERPLib* is adding and removing objects using *Python* operators += and −=. As usual, you can add an ID, a record, or a list of them:

With a list of records:

```
>>> user = oerp.get('res.users').browse(1)
>>> groups = oerp.get('res.groups').browse([4, 5])
>>> user.groups_id += list(groups)
>>> [g.id for g in user.groups_id]
[1, 2, 3, 4, 5]
```

With a list of record IDs:

```
>>> user.groups_id += [4, 5]
>>> [g.id for g in user.groups_id]
[1, 2, 3, 4, 5]
```

With an ID only:

```
>>> user.groups_id -= 4
>>> [g.id for g in user.groups_id]
[1, 2, 3, 5]
```

With a record only:

```
>>> group = oerp.get('res.groups').browse(5)
>>> user.groups_id -= group
>>> [g.id for g in user.groups_id]
[1, 2, 3]
```

### Reference

To update a `reference` field, you have to use either a string or a browsable record as below:

```
>>> helpdesk = oerp.browse('crm.helpdesk', 1)
>>> helpdesk.ref = 'res.partner,1' # with a string with the format '{relation},{id}'
>>> oerp.write_record(helpdesk)
>>> partner = oerp.browse('res.partner', 1)
>>> helpdesk.ref = partner # with a browsable record
>>> oerp.write_record(helpdesk)
```

A check is made on the relation name:

```
>>> helpdesk.ref = 'foo.bar,42'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "oerplib/service/osv/fields.py", line 213, in __set__
    value = self.check_value(value)
  File "oerplib/service/osv/fields.py", line 244, in check_value
    self._check_relation(relation)
  File "oerplib/service/osv/fields.py", line 225, in _check_relation
    field_name=self.name,
ValueError: The value 'foo.bar' supplied doesn't match with the possible values '['res.partner', 'cal
```

### Date and Datetime

`date` and `datetime` fields accept either string values or `datetime.date`/`datetime.datetime` objects.

With `datetime.date` and `datetime.datetime` objects:

```
>>> import datetime
>>> order = oerp.browse('purchase.order', 42)
>>> order.date_order = datetime.date(2011, 9, 20)
>>> order.minimum_planned_date = datetime.datetime(2011, 9, 20, 12, 31, 24)
>>> oerp.write_record(order)
```

With formated strings:

```
>>> order.date_order = "2011-09-20"                      # %Y-%m-%d
>>> order.minimum_planned_date = "2011-09-20 12:31:24"   # %Y-%m-%d %H:%M:%S
>>> oerp.write_record(order)
```

As always, a wrong type will raise an exception:

```
>>> order.date_order = "foobar"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "oerplib/fields.py", line 187, in setter
    value = self.check_value(value)
  File "oerplib/fields.py", line 203, in check_value
    self.pattern))
ValueError: Value not well formatted, expecting '%Y-%m-%d' format
```

### 3.2.5 Generate reports

Another nice functionnality is the reports generation (related to the `/report` *RPC* service) with the `report` method. You have to supply the name of the report, the name of the model and the ID of the record related:

```
>>> oerp.report('sale.order', 'sale.order', 1)
'/tmp/oerplib_uJ8Iho.pdf'
>>> oerp.report('webkitaccount.invoice', 'account.invoice', 1)
'/tmp/oerplib_r1W9jG.pdf'
```

The method will return the path to the generated temporary report file.

### 3.2.6 Manage databases

You can manage *OpenERP* databases with the `oerplib.OERP.db` property. It offers you a dynamic access to all methods of the `/db` RPC service in order to list, create, drop, dump, restore databases and so on.

---

**Note:** You have not to be logged in to perform database management tasks. Instead, you have to use the "super admin" password.

---

Prepare a connection:

```
>>> import oerplib
>>> oerp = oerplib.OERP(server='localhost')
```

At this point, you are able to list databases of this server:

```
>>> oerp.db.list()
[]
```

Let's create a new database:

```
>>> database_id = oerp.db.create('super_admin_passwd', 'test_db', False, 'fr_FR', 'admin')
```

The creation process may take some time on the *OpenERP* server, and you have to wait before using the new database. The state of the creation process is returned by the `get_progress` method:

```
>>> database_id = oerp.db.create('super_admin_passwd', 'test_db', False, 'fr_FR', 'admin')
>>> while not oerp.db.get_progress('super_admin_passwd', database_id)[0]
...     pass
>>> oerp.login('admin', 'admin', 'test_db')
```

However, *OERPLib* simplifies this by providing the `create_and_wait` method:

```
>>> oerp.db.create_and_wait('super_admin_passwd', 'test_db', False, 'fr_FR', 'admin')
[{'login': u'admin', 'password': u'admin', 'name': u'Administrator'},
 {'login': u'demo', 'password': u'demo', 'name': u'Demo User'}]
```

Some documentation about methods offered by the *OpenERP* `/db` RPC service is available `here`.

### 3.2.7 Inspect the metadata of OpenERP (New in version 0.8)

#### Draw a graph of relationships between models

**Note:** This functionality requires the installation of pydot.

The `relations` method will help you to generate a graphic of such relationships:

```
>>> graph = oerp.inspect.relations(['res.partner'])
>>> graph.write('rel_res_partner_v1.png', format='png')
```

By default, only the direct relationships of the model `res.partner` are shown (this behaviour can be changed with the `maxdepth` parameter), and model attributes are hidden. You can control the displayed models through the `whitelist` and `blacklist` parameters. For instance, assume that you only want data models whose name begins with *res.partner* excluding the *res.partner.bank* model:

```
>>> graph = oerp.inspect.relations(['res.partner'], whitelist=['res.partner*'], blacklist=['res.partn
>>> graph.write('rel_res_partner_v2.png', format='png')
```

**Note:** The blacklist has a higher priority than the whitelist

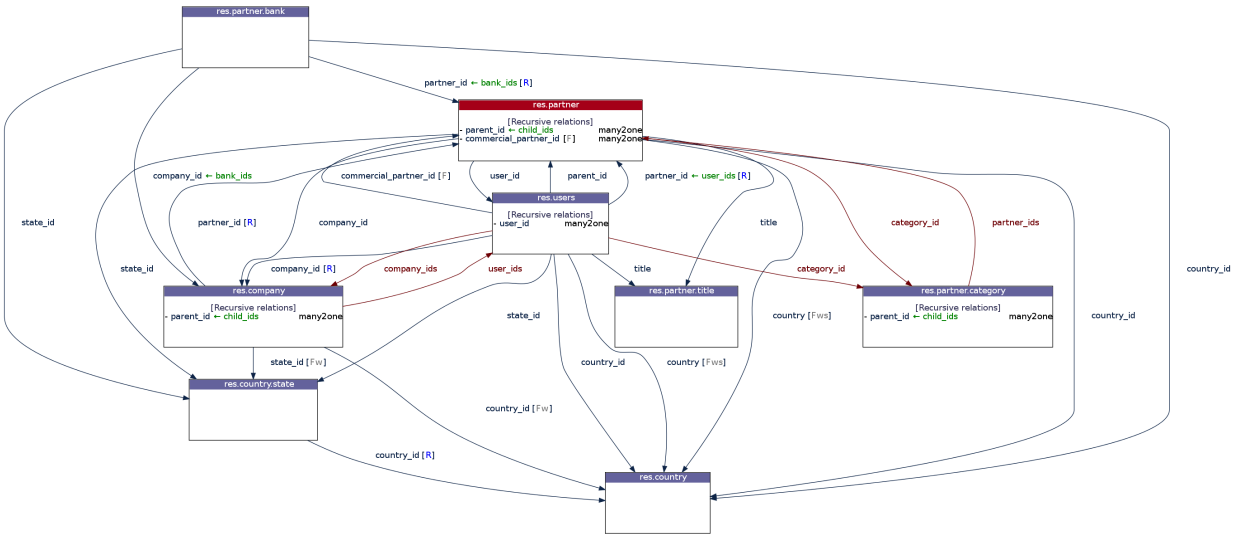
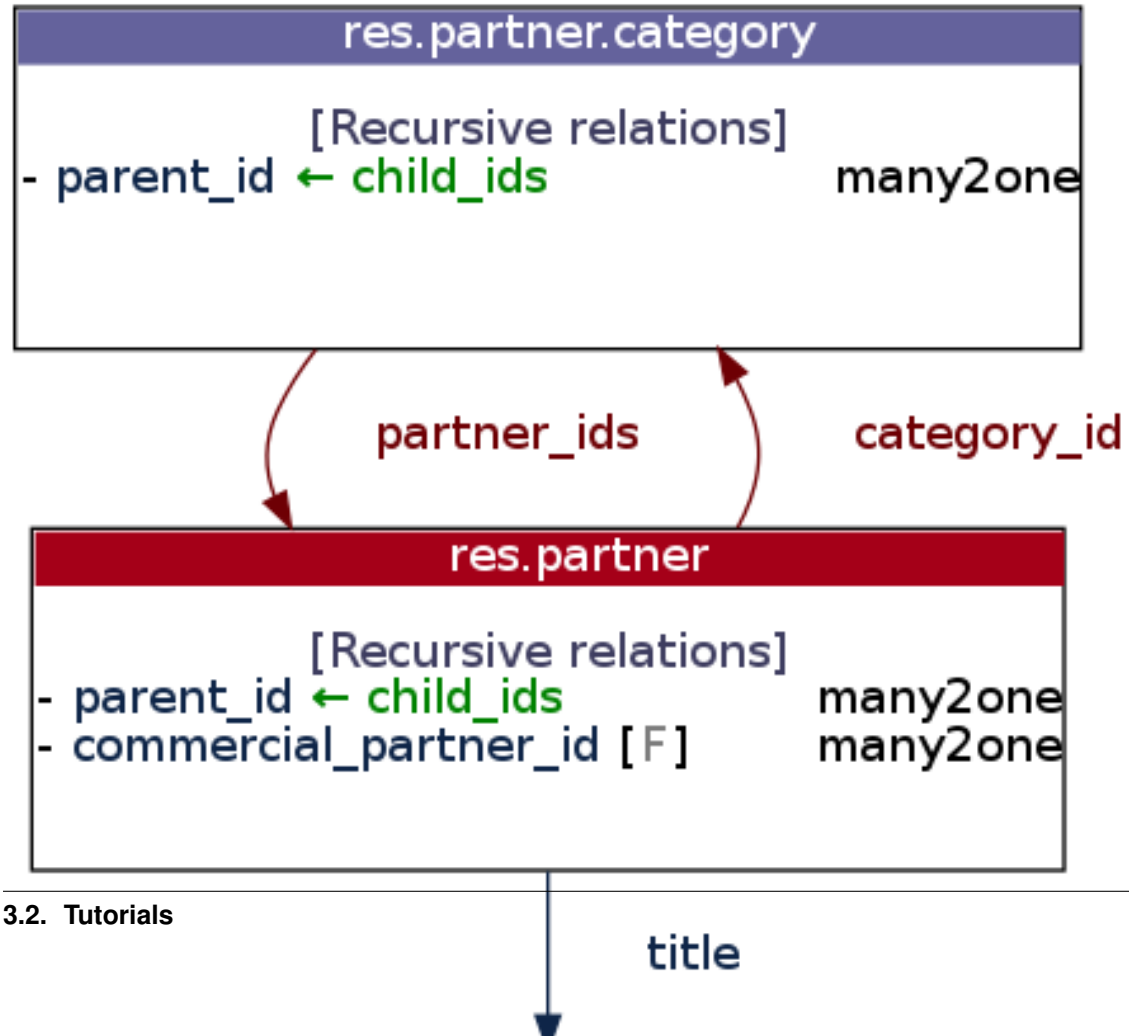


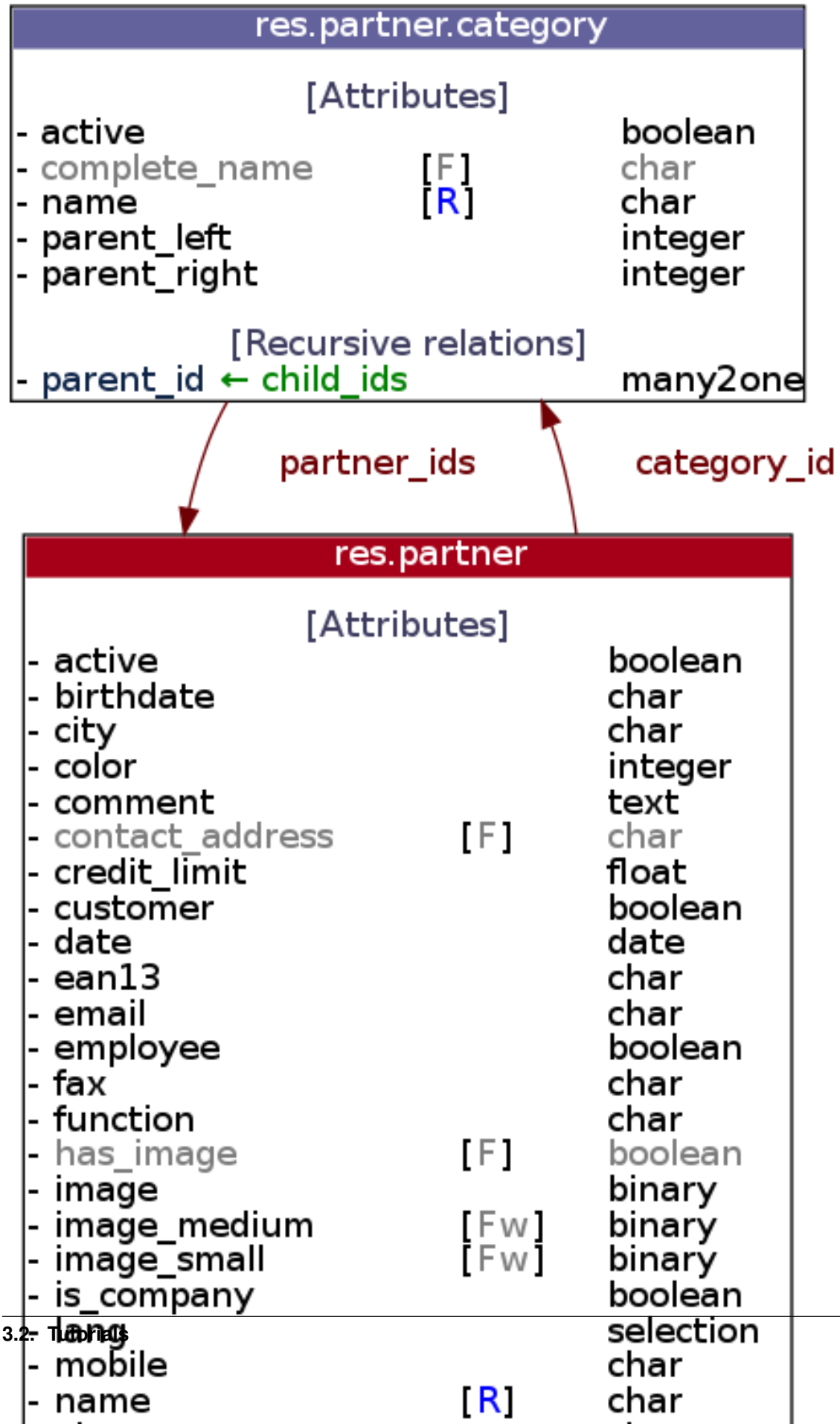

Figure 3.1: Legend:

| Element | Meaning |
|---|---|
| | many2one |
| | one2many |
| | many2many |
| | Field required |
| | Field function (readonly) |
| | Field function (writable) |
| | Field function (searchable) |



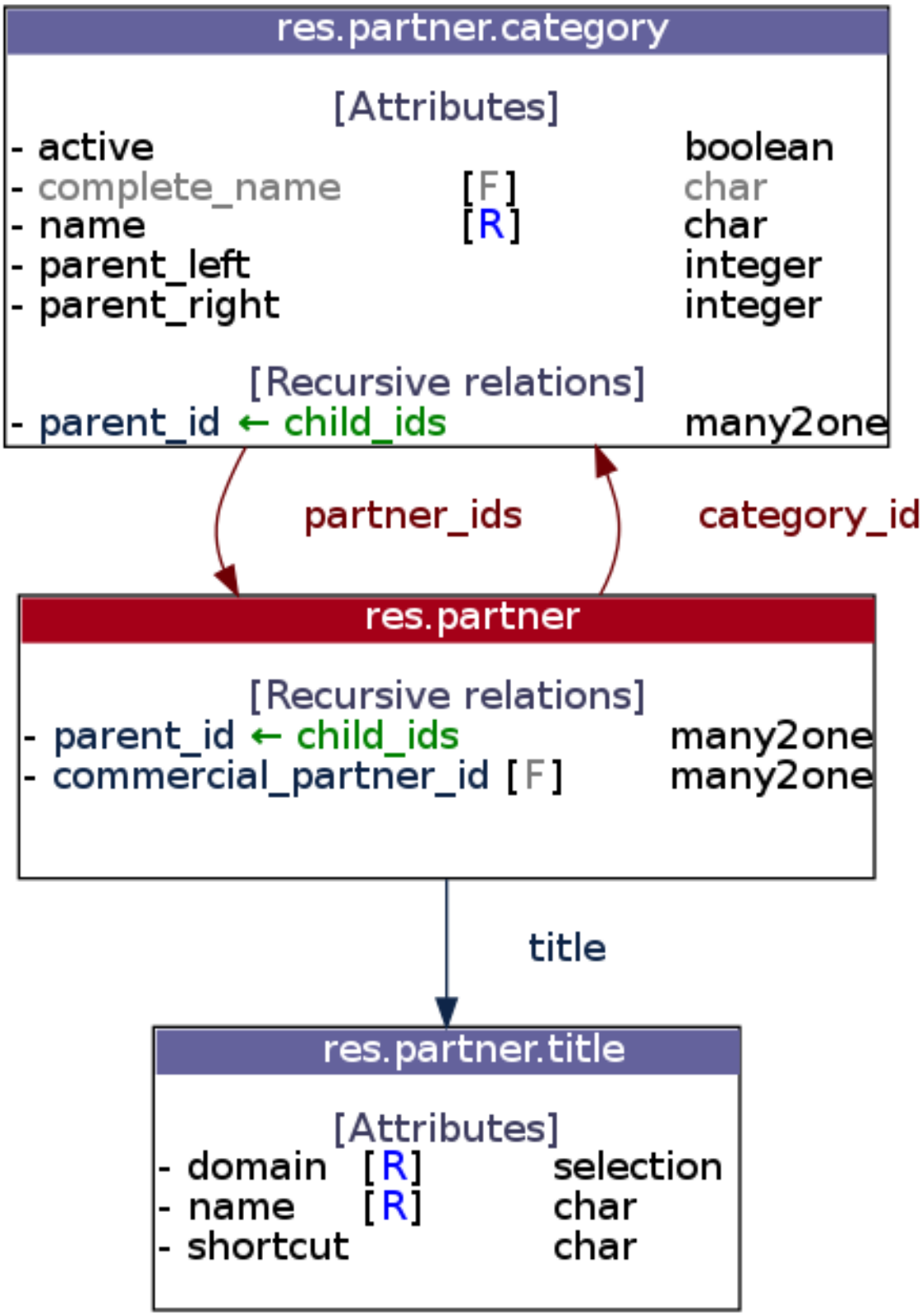
To display attributes, use the `attrs_whitelist` parameter. A wildcard is used here to show attributes of all models (but you can specify which models you want):

```
>>> graph = oerp.inspect.relations(['res.partner'], whitelist=['res.partner*'], blacklist=['res.partn
>>> graph.write('rel_res_partner_v3.png', format='png')
```


res.partner.category
[Attributes]
- active boolean
- complete_name [F] char
- name [R] char
- parent_left integer
- parent_right integer
[Recursive relations]
- parent_id ← child_ids many2one
partner_ids
category_id
res.partner
[Attributes]
- active boolean
- birthdate char
- city char
- color integer
- comment text
- contact_address [F] char
- credit_limit float
- customer boolean
- date date
- ean13 char
- email char
- employee boolean
- fax char
- function char
- has_image [F] boolean
- image binary
- image_medium [Fw] binary
- image_small [Fw] binary
- is_company boolean
- lang selection
- mobile char
- name [R] char

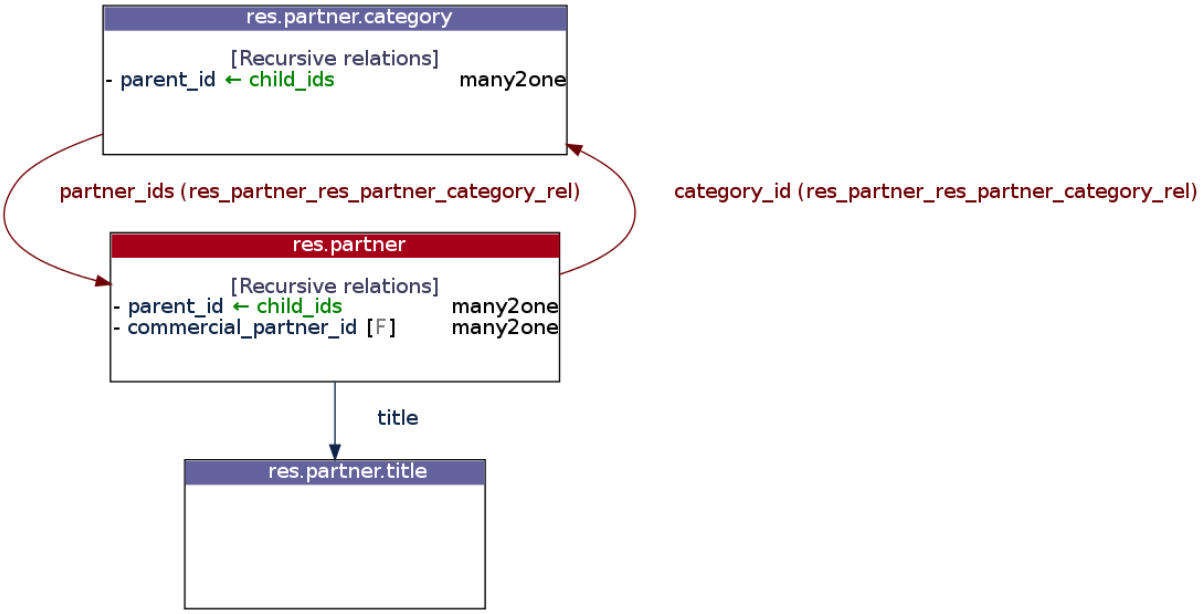
To hide attributes of some models, you can use the `attrs_blacklist` parameter:

```
>>> graph = oerp.inspect.relations(['res.partner'], whitelist=['res.partner*'], blacklist=['res.partn
>>> graph.write('rel_res_partner_v4.png', format='png')
```



Also, some configuration options can be set through the *config* parameter. Here is how to display *many2many* table names:

```
>>> config = {'show_many2many_table': True}
>>> graph = oerp.inspect.relations(['res.partner'], whitelist=['res.partner*'], blacklist=['res.partn
>>> graph.write('rel_res_partner_v5.png', format='png')
```



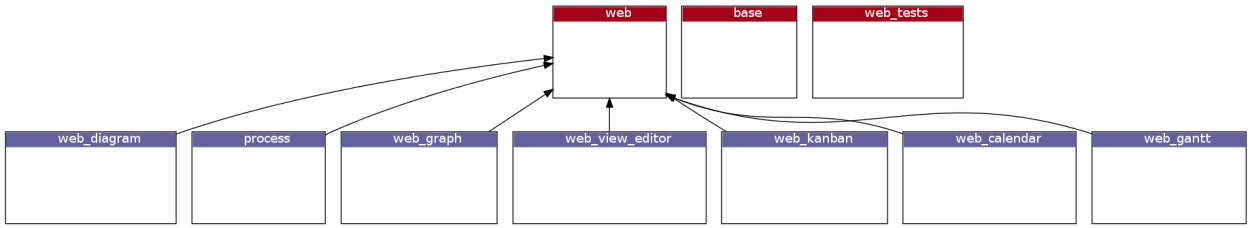

For more details, take a look at the `relations` method documentation.

### Draw a graph of module dependencies

**Note:** This functionality requires the installation of pydot.

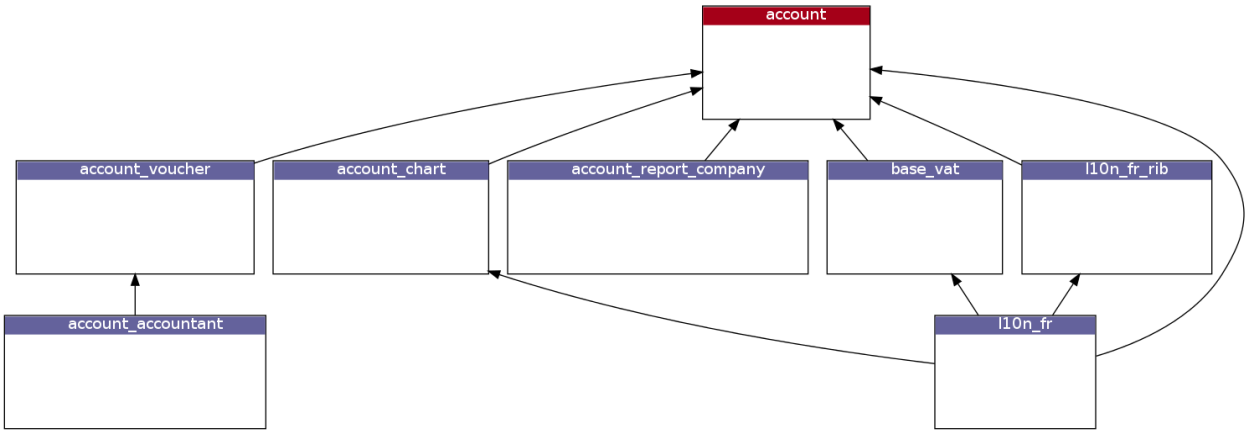
You will be able to generate a graphic representing dependencies between all modules with the `dependencies` method:

```
>>> graph = oerp.inspect.dependencies()
>>> graph.write('dependencies_v1.png', format='png')
```



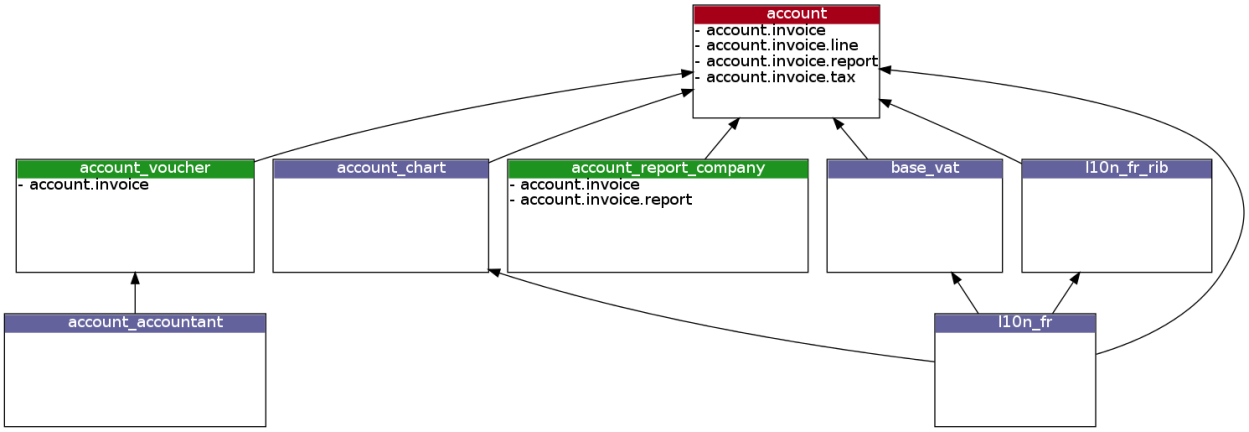

By default all installed modules are shown on the resulting graph, the red ones can be seen as *root* modules (they depend on no module in the current graph). Assume we have installed the *Accounting and Finance* application, and want to only display dependencies related to the *account* module:

```
>>> graph = oerp.inspect.dependencies(['account'])
>>> graph.write('dependencies_v2.png', format='png')
```




This time the *root* module is `account`. Modules may also contain data models. To highlight some of them among the modules, set the *models* and *models_blacklist* parameters with one or several patterns (a joker * can be used):

```
>>> graph = oerp.inspect.dependencies(['account'], models=['account.invoice.*'])
>>> graph.write('dependencies_v3.png', format='png')
```

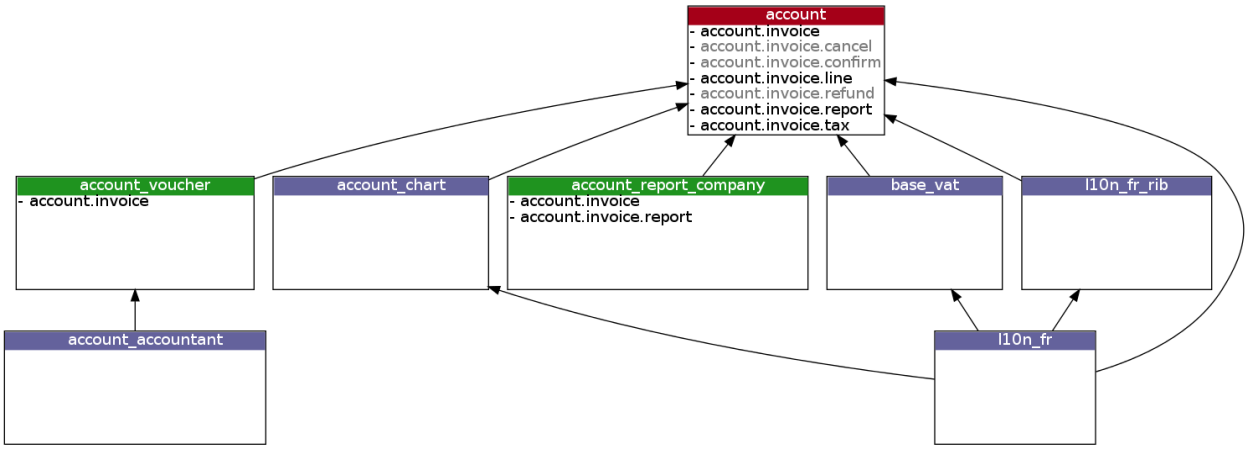


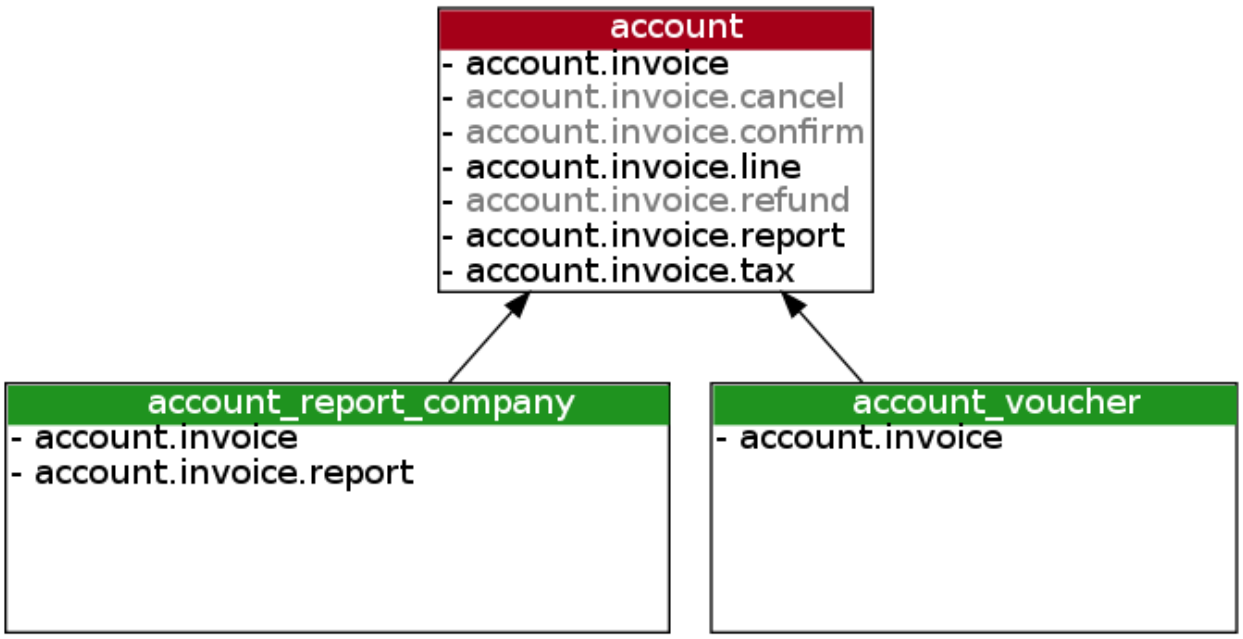

Modules related to the matching models are shown in green (in addition to the red one). It is possible to display transient models too through the `show_transient_model` configuration option (displayed in gray in the following graph):

```
>>> config = {'show_transient_model': True}
>>> graph = oerp.inspect.dependencies(['account'], models=['account.invoice.*'], config=config)
>>> graph.write('dependencies_v4.png', format='png')
```

To hide "noisy" modules and restrict the resulting graph only to data models that interest you, add the `restrict=True` parameter:

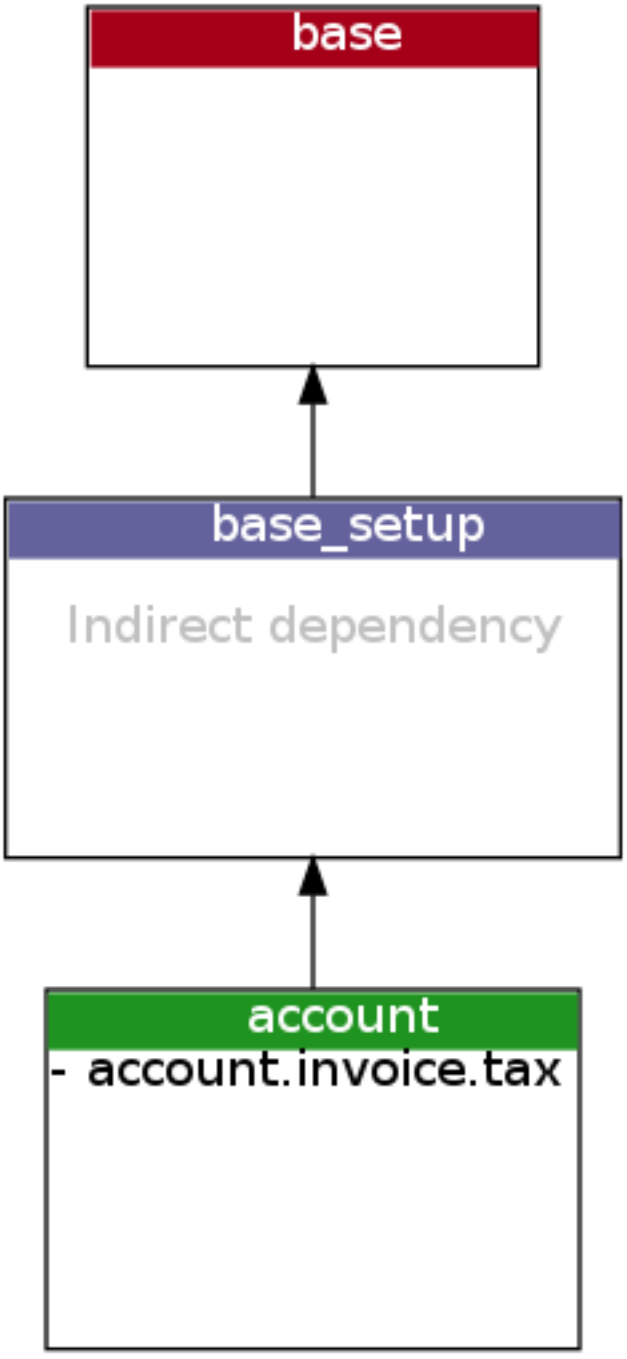
```
>>> config = {'show_transient_model': True}
>>> graph = oerp.inspect.dependencies(['account'], ['account.invoice.*'], restrict=True, config=confi
>>> graph.write('dependencies_v5.png', format='png')
```




Even in restricted mode, *root* modules which are not concerned by matching *models* are always displayed. Also, if no dependency can be satisfied between modules, the method will try to add one. For instance, the *base* module have no `account.invoice.tax` model, and *account* does not directly depends on *base*, so a dependency between *base* and *account* should be added to display a suitable graph:

```
>>> graph = oerp.inspect.dependencies(['base'], ['account.invoice.tax'], restrict=True)
>>> graph.write('dependencies_v6.png', format='png')
```




For more details, take a look at the `dependencies` method documentation.

### Scan the views of data models to list *on_change* methods

*on_change* functions of a model can be listed with the `scan_on_change` method. Each detected function can be present on several views:

```
>>> oerp.inspect.scan_on_change(['res.users'])
{'res.users': {'on_change_company_id': {'base.view_users_form_simple_modif': {'company_id': ['company
                                        'mail.view_users_form_simple_modif_mail': {'company_id': ['co
```

```
'onchange_state': {'base.view_users_simple_form': {'state_id': ['state_id']}}}}
```

The dictionary returned is formatted as follows: `{model: {on_change: {view_name: field: [args]}}}`, e.g. the `onchange_state` method is set on the `state_id` field of the view `base.view_users_simple_form`, and take the same field as parameter.

### 3.2.8 Save the session to open it quickly later (New in version 0.8)

Once you are authenticated with your `OERP` instance, you can `save` these connection information under a code name and use this one to quickly instanciate a new `OERP` class:

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> user = oerp.login('admin', 'admin', 'my_database')
>>> oerp.save('foo')
```

By default, these informations are stored in the `~/.oerplibrc` file. You can however use another file:

```
>>> oerp.save('foo', '~/my_own_oerplibrc')
```

Then, use the `oerplib.OERP.load()` class method:

```
>>> import oerplib
>>> oerp = oerplib.OERP.load('foo')
```

Or, if you have saved your configuration in another file:

```
>>> oerp = oerplib.OERP.load('foo', '~/my_own_oerplibrc')
```

You can check available sessions with `oerplib.OERP.list()`, and remove them with `oerplib.OERP.remove()`:

```
>>> oerplib.OERP.list()
['foo']
>>> oerplib.OERP.remove('foo')
>>> 'foo' not in oerplib.OERP.list()
True
```

### 3.2.9 Change the timeout

By default, the timeout is set to 120 seconds for all RPC requests. If your requests need a higher timeout, you can set it through the `oerplib.OERP.config` property:

```
>>> oerp.config['timeout']
120
>>> oerp.config['timeout'] = 300  # Set the timeout to 300 seconds
```

## 3.3 Frequently Asked Questions (FAQ) (New in version 0.8)

### 3.3.1 Connect to an OpenERP Online (SaaS) instance

First, you have to connect on your *OpenERP* instance, and set a password for your user account in order to active the *XML-RPC* protocol.

Then, just use the `xmlrpc+ssl` protocol with the port 443:

```
>>> import oerplib
>>> oerp = oerplib.OERP('foobar.my.openerp.com', protocol='xmlrpc+ssl', port=443)
>>> oerp.db.server_version()
'7.saas~1'
```

### 3.3.2 Update a record with an *on_change* method

*OERPLib* does not provide helpers for such methods currently. A call to an `on_change` method intend to be executed from a view and there is no support for that (not yet?) such as fill a form, validate it, etc...

But you can emulate an `on_change` by writing your own function, for instance:

```
def on_change(oerp, record, method, *args):
    """Update `record` with the result of the on_change `method`"""
    res = oerp.execute(record.__osv__['name'], method, *args)
    for k, v in res['value'].iteritems():
        setattr(record, k, v)
    return record
```

And call it on a record with the desired method and its parameters:

```
>>> order = oerp.get('sale.order').browse(42)
>>> order = on_change(oerp, order, 'product_id_change', ARGS...)
>>> oerp.write_record(order)  # Save your record
```

To know what parameters to send to the `on_change`, the `scan_on_change` method can help you.

### 3.3.3 Some OSV methods does not accept the *context* parameter

Since *OpenERP* 6.1, the `context` parameter can be sent automatically for each call to an *OSV/Model* method (this is the default behaviour since *OERPLib* 0.7). But on the side of the *OpenERP* server, some *OSV* methods have no `context` parameter, and *OERPLib* has no way to guess it, which results in an nasty exception. So you have to disable temporarily this behaviour by yourself by setting the `auto_context` option to `False`:

```
>>> oerp.config['auto_context'] = False  # 'get()' method of 'ir.sequence' does not support the conte
>>> next_seq = oerp.get('ir.sequence').get('stock.lot.serial')
>>> oerp.config['auto_context'] = True  # Restore the configuration
```

### 3.3.4 Change the behaviour of a script according to the version of OpenERP

You can compare versions of *OpenERP* servers with the `v` function applied on the `OERP.version` property, for instance:

```
import oerplib
from oerplib.tools import v

for session in oerplib.OERP.list():
    oerp = oerplib.OERP.load(session)
    if v(oerp.version) <= v('6.1'):
        pass  # do some stuff
    else:
        pass  # do something else
```

## 3.4 Reference

### 3.4.1 Browse object fields

The table below presents the Python types returned by *OERPLib* for each *OpenERP* fields used by `browse_record` objects (see the `browse` method):

| *OpenERP* fields | Python types used in *OERPLib* |
|---|---|
| fields.binary | basestring (str or unicode) |
| fields.boolean | bool |
| fields.char | basestring (str or unicode) |
| fields.date | datetime.date |
| fields.datetime | datetime.datetime |
| fields.float | float |
| fields.integer | integer |
| fields.selection | basestring (str or unicode) |
| fields.text | basestring (str or unicode) |

Exceptions made for relation fields:

| *OpenERP* fields | Types used in *OERPLib* |
|---|---|
| fields.many2one | `browse_record` instance |
| fields.one2many | generator to iterate on `browse_record` instances |
| fields.many2many | generator to iterate on `browse_record` instances |
| fields.reference | `browse_record` instance |

### 3.4.2 oerplib

The *oerplib* module defines the `OERP` class.

The `OERP` class manage the client-side operations which are related to an *OpenERP* server. You can use this one to write *Python* programs that performs a variety of automated jobs that communicate with an *OpenERP* server.

You can load a pre-configured `OERP` session with the `load()` function.

Here's a sample session using this module:

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')                    # connect to localhost, default port
>>> user = oerp.login('admin', 'admin', 'my_database')  # login returns an user object
>>> user.name
'Administrator'
>>> oerp.save('foo')                                    # save session informations in ~/.oerplibrc
>>> oerp = oerplib.load('foo')                          # get a pre-configured session from ~/.oerpl
```

### 3.4.3 oerplib.OERP

**class** `oerplib.`**`OERP`**(*server='localhost'*, *database=None*, *protocol='xmlrpc'*, *port=8069*, *timeout=120*, *version=None*)

Return a new instance of the `OERP` class. The optional *database* parameter specifies the default database to use when the `login` method is called. If no *database* is set, the *database* parameter of the `login` method will be mandatory.

*XML-RPC* and *Net-RPC* protocols are supported. Respective values for the *protocol* parameter are `xmlrpc`, `xmlrpc+ssl` and `netrpc`.

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost', protocol='xmlrpc', port=8069)
```

Since the version *0.7*, *OERPLib* will try by default to detect the *OpenERP* server version in order to adapt its requests. However, it is possible to force the version of *OpenERP* with the *version* parameter:

```
>>> oerp = oerplib.OERP('localhost', version='6.0')
```

**Raise** `oerplib.error.InternalError`, `oerplib.error.RPCError`

**browse**(*model*, *ids*, *context=None*)

Browse one or several records (if *ids* is a list of IDs) from *model*. The fields and values for such objects are generated dynamically.

```
>>> oerp.browse('res.partner', 1)
browse_record(res.partner, 1)

>>> [partner.name for partner in oerp.browse('res.partner', [1, 2])]
[u'Your Company', u'ASUStek']
```

A list of data types used by `browse_record` fields are available *here*.

**Returns** a `browse_record` instance

**Returns** a generator to iterate on `browse_record` instances

**Raise** `oerplib.error.RPCError`

**common**

New in version 0.6.

The common service (`/common` RPC service). See the `oerplib.service.common.Common` class.

**config**

Dictionary of available configuration options.

```
>>> oerp.config
{'auto_context': True, 'timeout': 120}
```

- `auto_context`: if set to *True*, the user context will be sent automatically to every call of a `model` method (default: *True*):

  New in version 0.7.

  **Note:** This option only works on *OpenERP* version *6.1* and above.

  ```
  >>> product_osv = oerp.get('product.product')
  >>> product_osv.name_get([3]) # Context sent by default ('lang': 'fr_FR' here)
  [[3, '[PC1] PC Basic']]
  >>> oerp.config['auto_context'] = False
  >>> product_osv.name_get([3]) # No context sent
  [[3, '[PC1] Basic PC']]
  ```

- `timeout`: set the maximum timeout in seconds for a RPC request (default: *120*):

  New in version 0.6.

  ```
  >>> oerp.config['timeout'] = 300
  ```

**context**
The context of the user connected.

```
>>> oerp.login('admin', 'admin')
browse_record('res.users', 1)
>>> oerp.context
{'lang': 'fr_FR', 'tz': False}
>>> oerp.context['lang'] = 'en_US'
```

**create**(*model*, *vals*, *context=None*)
Create a new *model* record with values contained in the *vals* dictionary.

```
>>> partner_id = oerp.create('res.partner', {'name': 'Jacky Bob', 'lang': 'fr_FR'})
```

**Returns** the ID of the new record.

**Raise** `oerplib.error.RPCError`

**database**
The database currently used.

**db**
New in version 0.4.

The database management service (`/db` RPC service). See the `oerplib.service.db.DB` class.

**exec_workflow**(*model*, *signal*, *obj_id*)
Execute the workflow *signal* on the instance having the ID *obj_id* of *model*.

**Raise** `oerplib.error.RPCError`

**execute**(*model*, *method*, **args*)
Execute the *method* of *model*. **args* parameters varies according to the *method* used.

```
>>> oerp.execute('res.partner', 'read', [1, 2], ['name'])
[{'name': u'ASUStek', 'id': 2}, {'name': u'Your Company', 'id': 1}]
```

**Returns** the result returned by the *method* called

**Raise** `oerplib.error.RPCError`

**execute_kw**(*model*, *method*, *args=None*, *kwargs=None*)
Execute the *method* of *model*. *args* is a list of parameters (in the right order), and *kwargs* a dictionary (named parameters). Both varies according to the *method* used.

```
>>> oerp.execute_kw('res.partner', 'read', [[1, 2]], {'fields': ['name']})
[{'name': u'ASUStek', 'id': 2}, {'name': u'Your Company', 'id': 1}]
```

**Warning:** This method only works on *OpenERP* version *6.1* and above.

**Returns** the result returned by the *method* called

**Raise** `oerplib.error.RPCError`

**get**(*model*)
New in version 0.5.

Return a proxy of the *model* built from the *OpenERP* server (see `oerplib.service.osv.Model`).

**Returns** an instance of `oerplib.service.osv.Model`

**static get_osv_name**(*browse_record*)
Deprecated since version 0.7: use the __osv__ attribute instead (see BrowseRecord).

```
>>> partner = oerp.browse('res.partner', 1)
>>> oerp.get_osv_name(partner)
'res.partner'
```

**Returns** the model name of the browsable record

**inspect**
New in version 0.8.

The inspect service (custom service). See the oerplib.service.inspect.Inspect class.

**classmethod list**(*rc_file='~/.oerplibrc'*)
New in version 0.8.

Return a list of all sessions available in the *rc_file* file:

```
>>> import oerplib
>>> oerplib.OERP.list()
['foo', 'bar']
```

Then, use the load() function with the desired session:

```
>>> oerp = oerplib.OERP.load('foo')
```

**classmethod load**(*name*, *rc_file='~/.oerplibrc'*)
New in version 0.8.

Return a OERP session pre-configured and connected with informations identified by *name*:

```
>>> import oerplib
>>> oerp = oerplib.OERP.load('foo')
```

Such informations are stored with the OERP.save method.

**login**(*user='admin'*, *passwd='admin'*, *database=None*)
Log in as the given *user* with the password *passwd* on the database *database* and return the corresponding user as a browsable record (from the res.users model). If *database* is not specified, the default one will be used instead.

```
>>> user = oerp.login('admin', 'admin', database='db_name')
>>> user.name
u'Administrator'
```

**Returns** the user connected as a browsable record

**Raise** oerplib.error.RPCError, oerplib.error.Error

**port**
The port used.

**protocol**
The protocol used.

**read**(*model*, *ids*, *fields=None*, *context=None*)
Return *fields* values for each *model* record identified by *ids*. If *fields* is not specified, all fields values will be retrieved.

```
>>> oerp.read('res.partner', [1, 2], ['name'])
[{'name': u'ASUStek', 'id': 2}, {'name': u'Your Company', 'id': 1}]
```

**Returns** list of dictionaries

**Raise** `oerplib.error.RPCError`

**refresh** (*browse_record*, *context=None*)
Restore original values on *browse_record* with data fetched on the *OpenERP* database. As a result, all changes made locally on the record are canceled.

**Raise** `oerplib.error.RPCError`

**classmethod remove** (*name*, *rc_file='~/.oerplibrc'*)
New in version 0.8.

Remove the session identified by *name* from the *rc_file* file:

```
>>> import oerplib
>>> oerplib.OERP.remove('foo')
True
```

**report** (*report_name*, *model*, *obj_ids*, *report_type='pdf'*, *context=None*)
Download a report from the *OpenERP* server and return the path of the file.

```
>>> oerp.report('sale.order', 'sale.order', 1)
'/tmp/oerplib_uJ8Iho.pdf'
>>> oerp.report('sale.order', 'sale.order', [1, 2])
'/tmp/oerplib_giZS0v.pdf'
```

**Returns** the path to the generated temporary file

**Raise** `oerplib.error.RPCError`

**reset** (*browse_record*)
Cancel all changes made locally on the *browse_record*. No request to the server is executed to perform this operation. Therefore, values restored may be outdated.

**save** (*name*, *rc_file='~/.oerplibrc'*)
New in version 0.8.

Save the session configuration under the name *name*. These informations are stored in the `~/.oerplibrc` file by default.

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost', protocol='xmlrpc', port=8069)
>>> oerp.login('admin', 'admin', 'db_name')
>>> oerp.save('foo')
```

Such informations can be loaded with the `oerplib.load()` function by returning a pre-configured session of `OERP`, or with the *oerp* command line tool supplied with *oerplib*.

**search** (*model*, *args=None*, *offset=0*, *limit=None*, *order=None*, *context=None*, *count=False*)
Return a list of IDs of records matching the given criteria in *args* parameter. *args* must be of the form `[('name', '=', 'John'), (...)]`

```
>>> oerp.search('res.partner', [('name', 'like', 'Agrolait')])
[3]
```

**Returns** a list of IDs

**Raise** `oerplib.error.RPCError`

**server**
The server name.

**unlink**(*model*, *ids*, *context=None*)
Delete *model* records identified by *ids*.

```
>>> oerp.unlink('res.partner', [1])
```

**Returns** *True*

**Raise** `oerplib.error.RPCError`

**unlink_record**(*browse_record*, *context=None*)
New in version 0.4.

Delete the record corresponding to *browse_record* from the *OpenERP* database.

```
>>> partner = oerp.browse('res.partner', 1)
>>> oerp.unlink_record(partner)  # unlink('res.partner', [1])
```

**Returns** *True*

**Raise** `oerplib.error.RPCError`

**user**
The browsable record of the user connected.

```
>>> oerp.login('admin', 'admin') == oerp.user
True
```

**version**
The version of the OpenERP server.

```
>>> oerp.version
'7.0-20131014-231047'
```

**wizard**
New in version 0.6.

The wizard service (`/wizard` RPC service). See the `oerplib.service.wizard.Wizard` class.

**write**(*model*, *ids*, *vals=None*, *context=None*)
Update *model* records identified by *ids* with the given values contained in the *vals* dictionary.

```
>>> oerp.write('res.users', [1], {'name': "Administrator"})
True
```

**Returns** *True*

**Raise** `oerplib.error.RPCError`

**write_record**(*browse_record*, *context=None*)
New in version 0.4.

Update the record corresponding to *browse_record* by sending its values to the *OpenERP* database (only field values which have been changed).

```
>>> partner = oerp.browse('res.partner', 1)
>>> partner.name = "Test"
>>> oerp.write_record(partner)  # write('res.partner', [1], {'name': "Test"})
```

**Returns** *True*

**Raise** `oerplib.error.RPCError`

### 3.4.4 oerplib.service.osv

#### oerplib.service.osv.Model

**class** `oerplib.service.osv.`**`Model`**(*oerp*, *model*)

New in version 0.5.

Represent a data model from the *OpenERP* server.

---

**Note:** This class have to be used through the `oerplib.OERP.get()` method.

---

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> user = oerp.login('admin', 'passwd', 'database')
>>> user_obj = oerp.get('res.users')
>>> user_obj
<oerplib.service.osv.osv.Model object at 0xb75ba4ac>
>>> user_obj.name_get(user.id) # Use any methods from the model instance
[[1, 'Administrator']]
```

**Warning:** The only method implemented in this class is `browse`. Except this one, method calls are purely dynamic. As long as you know the signature of the model method targeted, you will be able to use it (see the *tutorial*).

**`browse`**(*ids*, *context=None*)

Browse one or several records (if *ids* is a list of IDs) from *model*. The fields and values for such objects are generated dynamically.

```
>>> oerp.get('res.partner').browse(1)
browse_record(res.partner, 1)
```

```
>>> [partner.name for partner in oerp.get('res.partner').browse([1, 2])]
[u'Your Company', u'ASUStek']
```

A list of data types used by `browse_record` fields are available *here*.

**Returns** a `browse_record` instance

**Returns** a generator to iterate on `browse_record` instances

**Raise** `oerplib.error.RPCError`

#### oerplib.service.osv.BrowseRecord

**class** `oerplib.service.osv.`**`BrowseRecord`**(*o_id*)

Base class that all browsable records inherit from. No attributes should be defined in this class (except `_id`/`id`,

__oerp__, __osv__, __data__ and Python magic methods) in order to not be conflicted with the fields defined in the model class on the *OpenERP* server.

A reference to the OERP object used to instanciate a browse_record is available through the __oerp__ attribute:

```
>>> oerp = oerplib.OERP('localhost')
>>> user = oerp.login('admin', 'admin', 'db_name')
>>> user.__oerp__ == oerp
True
```

The __data__ attribute is used to store some data related to the record (it is not recommended to edit them):

```
>>> user.__data__
{'updated_values': {},
 'raw_data': {'action_id': False,
              'active': True,
              'company_id': [1, 'Your Company'],
              ...},
 'values': {'action_id': False,
            'active': True,
            'company_id': [1, 'Your Company'],
            ...}}
```

In the same way, information about the model class and its columns may be obtained via the __osv__ attribute:

```
>>> user.__osv__
{'columns': {'action_id': <oerplib.service.osv.fields.Many2OneField object at 0xb75786ec>,
             'active': <oerplib.service.osv.fields.ValueField object at 0xb7598b6c>,
             'company_id': <oerplib.service.osv.fields.Many2OneField object at 0xb757868c>,
             ...},
 'name': 'res.users'}
```

**id**
ID of the record.

### 3.4.5 oerplib.service.common

**class** oerplib.service.common.**Common**(*oerp*)
New in version 0.6.

The *Common* class represents the /common RPC service. Lets you log in on *OpenERP*, and provides various utility functions.

---

**Note:** This service have to be used through the oerplib.OERP.common property.

---

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> oerp.common
<oerplib.service.common.Common object at 0xb76266ac>
```

**Warning:** All methods documented below are not strictly implemented in *OERPLib*
Method calls are purely dynamic, and the following documentation can be wrong if the API of *OpenERP* is changed between versions. Anyway, if you know the API used by the *OpenERP* server for the /common RPC service, it will work.

**login**(*db*, *login*, *password*)

```
>>> oerp.common.login('test_db', 'admin', 'admin_passwd')
1
```

**Returns** the user's ID or *False*

**authenticate**(*db*, *login*, *password*, *user_agent_env*)

```
>>> oerp.common.authenticate('test_db', 'admin', 'admin_passwd', {})
1
```

**Returns** the user's ID or *False*

**version**()

```
>>> oerp.common.version()
{'protocol_version': 1, 'server_version': '6.1'}
```

**about**(*extended=False*)

Return information about the *OpenERP* Server.

```
>>> oerp.common.about()
'\n\nOpenERP is an ERP+CRM program for small and medium businesses.\n\nThe whole source code
```

```
>>> oerp.common.about(True)
['\n\nOpenERP is an ERP+CRM program for small and medium businesses.\n\nThe whole source coc
```

**Param** extended: if *True* then return version info

**Returns** string if extended is *False* else tuple

**timezone_get**(*db*, *login*, *password*)

```
>>> oerp.common.timezone_get('test_db', 'admin', 'admin_passwd')
'UTC'
```

**get_server_environment**()

```
>>> print(oerp.common.get_server_environment())
Environment Information :
System : Linux-2.6.32-5-686-i686-with-debian-6.0.4
OS Name : posix
Distributor ID: Debian
Description:    Debian GNU/Linux 6.0.4 (squeeze)
Release:        6.0.4
Codename:       squeeze
Operating System Release : 2.6.32-5-686
Operating System Version : #1 SMP Mon Mar 26 05:20:33 UTC 2012
Operating System Architecture : 32bit
Operating System Locale : fr_FR.UTF8
Python Version : 2.6.6
OpenERP-Server Version : 5.0.16
Last revision No. & ID :
```

**login_message**()

```
>>> oerp.common.login_message()
'Welcome'
```

**set_loglevel**(*loglevel*, *logger=None*)

```
>>> oerp.common.set_loglevel('DEBUG')
```

**get_stats**()

```
>>> print(oerp.common.get_stats())
OpenERP server: 5 threads
Servers started
Net-RPC: running
```

**list_http_services**()

```
>>> oerp.common.list_http_services()
[]
```

**check_connectivity**()

```
>>> oerp.common.check_connectivity()
True
```

**get_os_time**()

```
>>> oerp.common.get_os_time()
(0.01, 0.0, 0.0, 0.0, 17873633.129999999)
```

**get_sqlcount**()

```
>>> oerp.common.get_sqlcount()
```

**get_available_updates**(*super_admin_password*, *contract_id*, *contract_password*)

```
>>> oerp.common.get_available_updates('super_admin_passwd', 'MY_CONTRACT_ID', 'MY_CONTRACT_P
```

**get_migration_scripts**(*super_admin_password*, *contract_id*, *contract_password*)

```
>>> oerp.common.get_migration_scripts('super_admin_passwd', 'MY_CONTRACT_ID', 'MY_CONTRACT_P
```

### 3.4.6 oerplib.service.db

**class** oerplib.service.db.**DB**(*oerp*)

New in version 0.4.

The *DB* class represents the database management service. It provides functionalities such as list, create, drop, dump and restore databases.

**Note:** This service have to be used through the `oerplib.OERP.db` property.

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> oerp.db
<oerplib.service.db.DB object at 0xb75fb04c>
```

**Warning:** All methods documented below are not strictly implemented in *OERPLib* (except the `create_and_wait` method).
Method calls are purely dynamic, and the following documentation can be wrong if the API of *OpenERP* is changed between versions. Anyway, if you know the API used by the *OpenERP* server for the `/db` RPC service, it will work.

**list**()
Return a list of the *OpenERP* databases:

```
>>> oerp.db.list()
['prod_db', 'test_db']
```

**Returns** a list of database names

**list_lang**()
Return a list of codes and names of language supported by *OpenERP*:

```
>>> oerp.db.list_lang()
[['sq_AL', 'Albanian / Shqipëri'], ['ar_AR', 'Arabic / '], ...]
```

**Returns** a list of pairs representing languages with their codes and names

**server_version**()
Return the version of the *OpenERP Server*:

```
>>> oerp.db.server_version()
'6.1'
```

**Returns** the version of the *OpenERP Server* as string

**dump**(*super_admin_passwd*, *database*)
Return a dump of *database* in *base64*:

```
>>> binary_data = oerp.db.dump('super_admin_passwd', 'prod_db')
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** the *base64* string representation of the *database*

**restore**(*super_admin_passwd*, *database*, *binary_data*)
Restore in *database* a dump previously created with the `dump` method:

```
>>> oerp.db.restore('super_admin_passwd', 'test_db', binary_data)
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**drop**(*super_admin_passwd*, *database*)
Drop the *database* from *OpenERP*:

```
>>> oerp.db.drop('super_admin_passwd', 'test_db')
True
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** *True*

**create**(*super_admin_passwd*, *database*, *demo_data=False*, *lang='en_US'*, *admin_passwd='admin'*)
Request the *OpenERP* server to create a new database named *database* which will have *admin_passwd* as administrator password and localized with the *lang* parameter. You have to set the flag *demo_data* to *True* in order to insert demonstration data.

As the creating process may take some time, you can execute the `get_progress` method with the database ID returned to know its current state.

```
>>> database_id = oerp.db.create('super_admin_passwd', 'test_db', False, 'fr_FR', 'my_admin_
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** the ID of the new database

**get_progress**(*super_admin_passwd*, *database_id*)
Check the state of the creating process for the database identified by the *database_id* parameter.

```
>>> oerp.db.get_progress('super_admin_passwd', database_id) # Just after the call to the 'cr
(0, [])
>>> oerp.db.get_progress('super_admin_passwd', database_id) # Once the database is fully cre
(1.0, [{'login': 'admin', 'password': 'admin', 'name': 'Administrator'},
       {'login': 'demo', 'password': 'demo', 'name': 'Demo User'}])
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** A tuple with the progressing state and a list of user accounts created (once the database is fully created).

**create_database**(*super_admin_passwd*, *database*, *demo_data=False*, *lang='en_US'*, *admin_passwd='admin'*)
*Available since OpenERP 6.1*

Similar to `create` but blocking.

```
>>> oerp.db.create_database('super_admin_passwd', 'test_db', False, 'fr_FR', 'my_admin_passw
True
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** *True*

**duplicate_database**(*super_admin_passwd*, *original_database*, *database*)
*Available since OpenERP 7.0*

Duplicate *original_database' as 'database*.

```
>>> oerp.db.duplicate_database('super_admin_passwd', 'prod_db', 'test_db')
True
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** *True*

**rename**(*super_admin_passwd*, *old_name*, *new_name*)
Rename the *old_name* database to *new_name*.

```
>>> oerp.db.rename('super_admin_passwd', 'test_db', 'test_db2')
True
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** *True*

**db_exist**(*database*)

Check if connection to database is possible.

```
>>> oerp.db.db_exist('prod_db')
True
```

**Returns** *True* or *False*

**change_admin_password**(*super_admin_passwd*, *new_passwd*)

Change the administrator password by *new_passwd*.

```
>>> oerp.db.change_admin_password('super_admin_passwd', 'new_passwd')
True
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** *True*

**create_and_wait**(*super_admin_passwd*, *database*, *demo_data=False*, *lang='en_US'*, *admin_passwd='admin'*)

---

**Note:** This method is not part of the official API of *OpenERP*. It's just a wrapper around the `create` and `get_progress` methods. For *OpenERP* in version *6.1* or above, please prefer the use of the standard `create_database` method.

---

Like the `create` method, but waits the end of the creating process by executing the `get_progress` method regularly to check its state.

```
>>> oerp.db.create_and_wait('super_admin_passwd', 'test_db', False, 'fr_FR', 'my_admin_passw
[{'login': 'admin', 'password': 'my_admin_passwd', 'name': 'Administrateur'},
 {'login': 'demo', 'password': 'demo', 'name': 'Demo User'}]
```

The super administrator password *super_admin_passwd* of *OpenERP* is required to perform this action.

**Returns** a list of user accounts created

**Raise** `oerplib.error.RPCError`

### 3.4.7 oerplib.service.wizard

**class** `oerplib.service.wizard.`**Wizard**(*oerp*)

New in version 0.6.

The *Wizard* class represents the `/wizard` RPC service which lets you access to the old-style wizards.

---

**Note:** This service have to be used through the `oerplib.OERP.wizard` property.

---

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> user = oerp.login('admin', 'passwd', 'database')
```

```
>>> oerp.wizard
<oerplib.service.wizard.Wizard object at 0xb76266ac>
```

> **Warning:** All methods documented below are not strictly implemented in *OERPLib*
> Method calls are purely dynamic, and the following documentation can be wrong if the API of *OpenERP* is changed between versions. Anyway, if you know the API used by the *OpenERP* server for the `/wizard` RPC service, it will work.

**create**(*wiz_name*, *datas=None*)

```
>>> oerp.wizard.create('wiz_name')
1
```

**Returns** the wizard's instance ID

**execute**(*wiz_id*, *datas*, *action='init'*, *context=None*)

```
>>> oerp.wizard.execute(1, {'one_field': "Value"})
```

### 3.4.8 oerplib.service.inspect (New in version 0.8)

Provide the `Inspect` class which can output useful data from *OpenERP*.

#### oerplib.service.inspect.Inspect

**class** `oerplib.service.inspect.`**Inspect**(*oerp*)

New in version 0.8.

The *Inspect* class provides methods to output useful data from *OpenERP*.

> **Note:** This service have to be used through the `oerplib.OERP.inspect` property.

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> user = oerp.login('admin', 'passwd', 'database')
>>> oerp.inspect
<oerplib.service.inspect.Inspect object at 0xb42fa84f>
```

**relations**(*models, maxdepth=1, whitelist=['*'], blacklist=[], attrs_whitelist=[], attrs_blacklist=[], config={}*)

Return a `Relations` object containing relations between data models, starting from *models* (depth = 0) and iterate recursively until reaching the *maxdepth* limit.

*whitelist* and *blacklist* of models can be defined with patterns (a joker `*` can be used to match several models like `account*`). The whitelist has a lower priority than the blacklist, and all models declared in *models* are automatically integrated to the *whitelist*.

In the same way, displaying attributes can be defined for each model with `attrs_whitelist` and `attrs_blacklist`. By default, model attributes are not displayed, unless the `'*'` pattern is supplied in `attrs_whitelist`, or if only the `attrs_blacklist` is defined.

```
>>> oerp.inspect.relations(
...     ['res.users'],
...     maxdepth=1,
...     whitelist=['res*'],
...     blacklist=['res.country*'],
...     attrs_whitelist=['*'],
...     attrs_blacklist=['res.partner', 'res.company'],
... ).write('res_users.png', format='png')
```

*config* is a dictionary of options to override some attributes of the graph. Here the list of options and their default values:

- relation_types: ['many2one', 'one2many', 'many2many'],
- show_many2many_table: False,
- color_many2one: #0E2548,
- color_one2many: #008200,
- color_many2many: #6E0004,
- model_root_bgcolor_title: #A50018,
- model_bgcolor_title: #64629C,
- model_color_title: white,
- model_color_subtitle': #3E3D60,
- model_bgcolor: white,
- color_normal: black,
- color_required: blue
- color_function: #D9602E
- space_between_models: 0.25,

```
>>> oerp.inspect.relations(
...     ['res.users'],
...     config={'relation_types': ['many2one']},  # Only show many2one relations
... ).write('res_users.png', format='png')
```

---

**Note:** With *OpenERP < 6.1*, *many2one* and *one2many* relationships can not be bound together. Hence, a *one2many* relationship based on a *many2one* will draw a separate arrow.

---

**dependencies** (*modules*=[], *models*=[], *models_blacklist*=[], *restrict=False*, *config={}*)

Return a `Dependencies` object describing dependencies between modules. The *modules* defines a list of root nodes to reach among all dependencies (modules not related to them are not displayed). The default behaviour is to compute all dependencies between installed modules. The *models* list can be used to display all matching models among computed dependencies.

*models* and *models_blacklist* parameters can be defined with patterns (a joker * can be used to match several models like `account*`). The whitelist (*models*) has a lower priority than the blacklist (*models_blacklist*):

```
>>> oerp.inspect.dependencies(
...     models=['res.partner*'],
...     models_blacklist=['res.partner.title', 'res.partner.bank'],
... ).write('dependencies_res_partner.png', format='png')
```

By default all installed modules are shown on the graph. To limit the result to modules related to the *base* one (its childs):

```
>>> oerp.inspect.dependencies(
...     ['base'],
...     ['res.partner*'],
...     ['res.partner.title', 'res.partner.bank'],
... ).write('dependencies_res_partner_base.png', format='png')
```

All modules related to *base* are shown on the resulting graph, and matching models are highlighted among them, but some modules remain empty. To hide these "noisy" modules and restrict the resulting graph to data models that interest you, add the `restrict=True` parameter:

```
>>> oerp.inspect.dependencies(
...     ['base'],
...     ['res.partner*'],
...     ['res.partner.title', 'res.partner.bank'],
...     restrict=True,
... ).write('dependencies_res_partner_base_restricted.png', format='png')
```

In any case, root *modules* are always displayed on the graph in restricted mode (even if they have no matching model), and some unrelated modules may be added to satisfy dependencies.

*config* is a dictionary of options to override some attributes of the graph. Here the list of options and their default values:

- module_uninst_bgcolor_title: #DEDFDE,
- module_uninst_color_title: black,
- module_inst_bgcolor_title: #64629C,
- module_inst_color_title: white,
- module_root_bgcolor_title: #A50018,
- module_root_color_title: white,
- module_highlight_bgcolor_title: #1F931F,
- module_highlight_color_title: white,
- module_bgcolor: white,
- module_color_comment: grey,
- model_color_normal: black,
- model_color_transient: #7D7D7D,
- show_module_inst: True,
- show_module_uninst: False,
- show_model_normal: True,
- show_model_transient: False,

```
>>> oerp.inspect.dependencies(
...     ['base'],
...     ['res.partner*'],
...     ['res.partner.title', 'res.partner.bank'],
...     config={'show_model_transient': True},  # Show TransientModel/osv_memory models
... ).write('dependencies_res_partner_transient.png', format='png')
```

**Note:** With *OpenERP 5.0*, data models can not be bound to their related modules, and as such the *models* and *models_blacklist* parameters are ignored.

**scan_on_change**(*models*)

Scan all *on_change* methods detected among views of *models*, and returns a dictionary formatted as `{model: {on_change: {view_id: field: [args]}}}`

```
>>> oerp.inspect.scan_on_change(['sale.order'])
{'sale.order': {
    'onchange_partner_id': {
        'sale.view_order_form': {
            'partner_id': ['partner_id']}},
    'onchange_partner_order_id': {
        'sale.view_order_form': {
            'partner_order_id': ['partner_order_id', 'partner_invoice_id', 'partner_shipping
    'onchange_pricelist_id': {
        'sale.view_order_form': {
            'pricelist_id': ['pricelist_id', 'order_line']}},
    'onchange_shop_id': {
        'sale.view_order_form': {
            'shop_id': ['shop_id']}},
    'shipping_policy_change': {
        'sale.view_order_form': {
            'order_policy': ['order_policy']}}},
 'sale.order.line': {
    'product_id_change': {
        'sale.view_order_form': {
            'product_id': [
                'parent.pricelist_id', 'product_id', 'product_uom_qty', 'product_uom',
                'product_uos_qty', 'product_uos', 'name', 'parent.partner_id', False, True,
                'parent.date_order', 'product_packaging', 'parent.fiscal_position', False, '
            'product_uom_qty': [
                'parent.pricelist_id', 'product_id', 'product_uom_qty', 'product_uom',
                'product_uos_qty', 'product_uos', 'name', 'parent.partner_id', False, False,
                'parent.date_order', 'product_packaging', 'parent.fiscal_position', True, 'c
    ...
 }}
```

### oerplib.service.inspect.Relations

**class** oerplib.service.inspect.relations.**Relations**(*oerp*, *models*, *maxdepth=1*, *whitelist=None*, *blacklist=None*, *attrs_whitelist=None*, *attrs_blacklist=None*, *config=None*)

Draw relations between models with *Graphviz*.

**make_dot**()

Returns a *pydot.Dot* object representing relations between models.

```
>>> graph = oerp.inspect.relations(['res.partner'])
>>> graph.make_dot()
<pydot.Dot object at 0x2bb0650>
```

See the pydot documentation for details.

**write**(**args*, ***kwargs*)

Write the resulting graph in a file. It is just a wrapper around the `pydot.Dot.write()` method (see

the pydot documentation for details). Below a common way to use it:

```
>>> graph = oerp.inspect.relations(['res.partner'])
>>> graph.write('relations_res_partner.png', format='png')
```

About supported formats, consult the Graphviz documentation.

#### oerplib.service.inspect.Dependencies

**class** oerplib.service.inspect.dependencies.**Dependencies**(*oerp*, *modules=None*, *models=None*, *models_blacklist=None*, *restrict=False*, *config=None*)

Draw dependencies between modules. Models can be displayed in their respecting modules as well.

**make_dot**()

Returns a *pydot.Dot* object representing dependencies between modules.

```
>>> graph = oerp.inspect.dependencies(['base'], ['res.partner'])
>>> graph.make_dot()
<pydot.Dot object at 0x2f01990>
```

See the pydot documentation for details.

**write**(**args*, ***kwargs*)

Write the resulting graph in a file. It is just a wrapper around the `pydot.Dot.write()` method (see the pydot documentation for details). Below a common way to use it:

```
>>> graph = oerp.inspect.dependencies(['base'], ['res.partner'])
>>> graph.write('dependencies_res_partner.png', format='png')
```

About supported formats, consult the Graphviz documentation.

### 3.4.9 oerplib.rpc

This module provides *RPC* connectors which use the *XML-RPC*, *Net-RPC* or *JSON-RPC* protocol to communicate with an *OpenERP* server.

Afterwards, *RPC* services and their associated methods can be accessed dynamically from the connector returned.

*XML-RPC* and *Net-RPC* provide the same interface, such as services like `db`, `common` or `object`. On the other hand, *JSON-RPC* provides a completely different interface, with services provided by Web modules of *OpenERP* like `web/session`, `web/dataset` and so on.

**class** oerplib.rpc.**Connector**(*server*, *port=8069*, *timeout=120*, *version=None*)

Connector base class defining the interface used to interact with an *OpenERP* server.

#### XML-RPC and Net-RPC connectors

**class** oerplib.rpc.**ConnectorXMLRPC**(*server*, *port=8069*, *timeout=120*, *version=None*)

Connector class using the *XML-RPC* protocol.

```
>>> from oerplib import rpc
>>> cnt = rpc.ConnectorXMLRPC('localhost', port=8069)
```

Login and retrieve ID of the user connected:

```
>>> uid = cnt.common.login('database', 'user', 'passwd')
```

Execute a query:

```
>>> res = cnt.object.execute('database', uid, 'passwd', 'res.partner', 'read', [1])
```

Execute a workflow query:

```
>>> res = cnt.object.exec_workflow('database', uid, 'passwd', 'sale.order', 'order_confirm', 4)
```

**class** oerplib.rpc.**ConnectorXMLRPCSSL**(*server*, *port=8069*, *timeout=120*, *version=None*)
Connector class using the *XML-RPC* protocol over *SSL*.

**class** oerplib.rpc.**ConnectorNetRPC**(*server*, *port=8070*, *timeout=120*, *version=None*)

**Note:** No longer available since *OpenERP 7.0*.

Connector class using the *Net-RPC* protocol.

**static** rpc.**get_connector**(*server*, *port=8069*, *protocol='xmlrpc'*, *timeout=120*, *version=None*)
Deprecated since version 0.8.

Return a *RPC* connector to interact with an *OpenERP* server. Supported protocols are:

- `xmlrpc`: Standard *XML-RPC* protocol (default),
- `xmlrpc+ssl`: *XML-RPC* protocol over *SSL*,
- `netrpc`: *Net-RPC* protocol made by *OpenERP* (no longer available since *OpenERP v7.0*).

If the *version* parameter is set to *None*, the last API supported will be used to send requests to *OpenERP*. Otherwise, you can force the API to use with the corresponding string version (e.g.: `'6.0'`, `'6.1'`, `'7.0'`, `...`):

```
>>> from oerplib import rpc
>>> cnt = rpc.get_connector('localhost', 8069, 'xmlrpc', version='7.0')
```

### JSON-RPC connectors (New in version 0.8)

**Warning:** The support of JSON-RPC is still in the experimental stage.

**class** oerplib.rpc.**ConnectorJSONRPC**(*server*, *port=8069*, *timeout=120*, *version=None*, *deserialize=True*)
Connector class using the *JSON-RPC* protocol.

```
>>> from oerplib import rpc
>>> cnt = rpc.ConnectorJSONRPC('localhost', port=8069)
```

Open a user session:

```
>>> cnt.proxy.web.session.authenticate(db='database', login='admin', password='admin')
{u'jsonrpc': u'2.0', u'id': 202516757,
 u'result': {u'username': u'admin', u'user_context': {u'lang': u'fr_FR', u'tz': u'Europe/Brussel
 u'db': u'test70', u'uid': 1, u'session_id': u'308816f081394a9c803613895b988540'}}
```

Read data of a partner:

```
>>> cnt.proxy.web.dataset.call(model='res.partner', method='read', args=[[1]])
{u'jsonrpc': u'2.0', u'id': 454236230,
 u'result': [{u'id': 1, u'comment': False, u'ean13': False, u'property_account_position': False,
```

You can send requests this way too:

```
>>> cnt.proxy['/web/dataset'].call(model='res.partner', method='read', args=[[1]])
{u'jsonrpc': u'2.0', u'id': 328686288,
 u'result': [{u'id': 1, u'comment': False, u'ean13': False, u'property_account_position': False,
```

Or like this:

```
>>> cnt.proxy['web']['dataset'].call(model='res.partner', method='read', args=[[1]])
{u'jsonrpc': u'2.0', u'id': 102320639,
 u'result': [{u'id': 1, u'comment': False, u'ean13': False, u'property_account_position': False,
```

**class** oerplib.rpc.**ConnectorJSONRPCSSL**(*server*, *port=8069*, *timeout=120*, *version=None*, *deserialize=True*)

Connector class using the *JSON-RPC* protocol over *SSL*.

```
>>> from oerplib import rpc
>>> cnt = rpc.ConnectorJSONRPCSSL('localhost', port=8069)
```

### 3.4.10 oerplib.tools

This module contains the Config class which manage the configuration related to an instance of OERP, and some useful helper functions used internally in *OERPLib*.

**class** oerplib.tools.**Config**(*oerp*, *options*)

Class which manage the configuration of an OERP instance.

---

**Note:** This class have to be used through the oerplib.OERP.config property.

---

```
>>> import oerplib
>>> oerp = oerplib.OERP('localhost')
>>> type(oerp.config)
<class 'oerplib.tools.Config'>
```

oerplib.tools.**clean_version**(*version*)

Clean a version string.

```
>>> from oerplib.tools import clean_version
>>> clean_version('7.0alpha-20121206-000102')
'7.0'
```

**Returns** a cleaner version string

oerplib.tools.**detect_version**(*server*, *protocol*, *port*, *timeout=120*)

Deprecated since version 0.8.

Try to detect the *OpenERP* server version.

```
>>> from oerplib.tools import detect_version
>>> detect_version('localhost', 'xmlrpc', 8069)
'7.0'
```

**Returns** the version as string

oerplib.tools.**v**(*version*)
Convert a version string to a tuple. The tuple can be use to compare versions between them.

```
>>> from oerplib.tools import v
>>> v('7.0')
[7, 0]
>>> v('6.1')
[6, 1]
>>> v('7.0') < v('6.1')
False
```

**Returns** the version as tuple

### 3.4.11 oerplib.tools.session (New in version 0.8)

This module contains some helper functions used to save and load sessions in *OERPLib*.

oerplib.tools.session.**get**(*name*, *rc_file='~/.oerplibrc'*)
Return the session configuration identified by *name* from the *rc_file* file.

```
>>> import oerplib
>>> oerplib.tools.session.get('foo')
{'protocol': 'xmlrpc', 'user': 'admin', 'timeout': 120, 'database': 'db_name', 'passwd': 'admin'
```

**Raise** oerplib.error.Error

oerplib.tools.session.**get_all**(*rc_file='~/.oerplibrc'*)
Return all session configurations from the *rc_file* file.

```
>>> import oerplib
>>> oerplib.tools.session.get_all()
{'foo': {'protocol': 'xmlrpc', 'user': 'admin', 'timeout': 120, 'database': 'db_name', 'passwd':
```

oerplib.tools.session.**remove**(*name*, *rc_file='~/.oerplibrc'*)
Remove the session configuration identified by *name* from the *rc_file* file.

```
>>> import oerplib
>>> oerplib.tools.session.remove('foo')
```

**Raise** oerplib.error.Error

oerplib.tools.session.**save**(*name*, *data*, *rc_file='~/.oerplibrc'*)
Save the *data* session configuration under the name *name* in the *rc_file* file.

```
>>> import oerplib
>>> oerplib.tools.session.save('foo', {'type': 'OERP', 'server': 'localhost', 'protocol': 'xmlrp
```

### 3.4.12 oerplib.error

This module contains all exceptions raised by *OERPLib* when an error occurred.

**exception** oerplib.error.**InternalError**(*message*, *oerp_traceback=False*)
Exception raised when an error occurred during an internal operation.

**exception** oerplib.error.**RPCError**(*message*, *oerp_traceback=False*)
Exception raised when an error related to a RPC query occurred.

CHAPTER 4

# Supported OpenERP versions

*OERPLib* has been tested on *OpenERP* server v5.0, v6.0, v6.1, v7.0 and v8.0. It should work on next versions if *OpenERP* keeps a stable API.

CHAPTER 5

# Supported Python versions

*OERPLib* support Python versions 2.6, 2.7.

CHAPTER 6

# License

This software is made available under the *LGPL v3* license.

CHAPTER 7

# Bugs or suggestions

Please, feel free to report bugs or suggestions in the Bug Tracker!

CHAPTER 8

# Make a donation

*OERPLib* is mainly developed on free time. To show your appreciation and support this project, it is possible to make a donation through *PayPal*:

CHAPTER 9

# Indices and tables

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

## O