odoo_rpc_client Documentation

Release 1.0.0

Dmytro Katyukha

Feb 21, 2019

Contents

Contents:

CHAPTER 1

Odoo RPC Client

Contents

1.1 Overview

This is core part of OpenERP Proxy

This project is just **RPC client** for Odoo. This project provides interface similar to Odoo internal code to perform operations on **Odoo** objects hiding **XML-RPC** or **JSON-RPC** behind.

1.1.1 Features

- *Python 3.3+* support
- You can call any public method on any OpenERP / Odoo object including: *read*, *search*, *write*, *unlink* and others
- Have *a lot of speed optimizations* (caching, read only requested fields, read data for all records in current set (cache), by one RPC call, etc)
- Desinged to take as more benefits of **IPython autocomplete** as posible

- Provides *browse_record* like interface, allowing to browse related models too. Supports *browse* method. Also adds method *search_records* to simplify search-and-read operations.
- *Extension support*. You can easily modify most of components of this lib creating Your own extensions and plugins. It is realy simple. See for examples in openerp_proxy/ext/ directory.
- *Plugin Support*. Plugins are same as extensions, but aimed to implement additional logic. For example look at odoo_rpc_client/plugins and odoo_rpc_client/plugin.py
- Support of **JSON-RPC** for *version 8+* of Odoo
- Support of using **named parametrs** in RPC method calls (server version 6.1 and higher).
- *Experimental* integration with AnyField
- Missed feature? fill and issue on GitHub or GitLab

1.1.2 Quick example

```
from odoo_rpc_client import Client

# assume that odoo server is listening localhost on standard 8069 port and
# have database 'my_db'.
client = Client('localhost', 'my_db', 'user', 'password')

# get current user
client.user
print(client.user.name)

# simple rpc calls
client.execute('res.partner', 'read', [user.partner_id.id])

# Model browsing
SaleOrder = client['sale.order']
s_orders = SaleOrder.search_records([])
for order in s_orders:
    print(order.name)
    for line in order.order_line:
        print("\t%s" % line.name)
    print("-" * 5)
    print()
```

1.1.3 Supported Odoo server versions

Tested with: - Odoo versions: *7.0*, *8.0*, *9.0*, *10.0*, *11.0*, *12.0* - Python versions: *2.7*, *3.3*, *3.4*, *3.5*, *3.6*, *3.7*

1.2 Install

This project is present on PyPI so it could be installed via PIP:

```
pip install odoo_rpc_client
```

1.3 Usage

1.3.1 Connect to server / database

The one diference betwen using as lib and using as shell is the way connection to database is created. When using as shell the primary object is session, which provides some interactivity. But when using as library in most cases there are no need for that interactivity, so connection should be created manualy, providing connection data from some other sources like config file or something else.

So here is a way to create connection

```
from odoo_rpc_client import Client
db = Client(host='my_host.int',
            dbname='my_db',
            user='my_db_user',
            pwd='my_password here')
```

And next all there same, no more differences betwen shell and lib usage.

1.3.2 General usage

For example lets try to find how many sale orders in 'done' state we have in our database. (Look above sections to get help on how to connect to Odoo database)

```
>>> sale_order_obj = db['sale.order']  # or You may use 'db.get_obj('sale.order')' if␣
↪You like
>>>
>>> # Now lets search for sale orders:
>>> sale_order_obj.search([('state', '=', 'done')], count=True)
5
```

So we have 5 orders in done state. So let's read them.

Default way to read data from Odoo is to search for required records with *search* method which return's list of IDs of records, then read data using *read* method. Both methods mostly same as Odoo internal ones:

```
>>> sale_order_ids = sale_order_obj.search([('state', '=', 'done')])
>>> sale_order_datas = sale_order_obj.read(sale_order_ids, ['name'])  # Last argument␣
↪is optional.
                                                                      # it describes␣
↪list of fields to read
                                                                      # if it is not␣
↪provided then all fields
                                                                      # will be read
>>> sale_order_datas[0]
{'id': 3,
 'name': 'SO0004'
}
```

As we see reading data in such way allows us to get list of dictionaries where each contain fields have been read

Another way to read data is to use *search_records* or *read_lecords* method. Each of these methods receives same aguments as `search` or `read` method respectively. But passing `count` argument for `search\_records` will cause error. Main difference betwen these methods in using *Record* class instead of *dict* for each record had been read. Record class provides some orm-like abilities for records, allowing for example access fields as attributes and provide mechanisms to lazily fetch related fields.

```
>>> sale_orders = sale_order_obj.search_records([('state', '=', 'done')])
>>> sale_orders[0]
R(sale.order, 9)[SO0011]
>>>
>>> # So we have list of Record objects. Let's check what they are
>>> so = sale_orders[0]
>>> so.id
9
>>> so.name
SO0011
>>> so.partner_id
R(res.partner, 9)[Better Corp]
>>>
>>> so.partner_id.name
Better Corp
>>> so.partner_id.active
True
```

1.4 Additional features

1.4.1 Plugins

In version 0.4 plugin system was completly refactored. At this version we start using extend_me library to build extensions and plugins easily.

Plugins are usual classes that provides functionality that should be available at `db.plugins.*` point, implementing logic not related to core system.

For more information see source code and documentation

CHAPTER 2

Odoo RPC Client Modules

2.1 odoo_rpc_client Package

2.1.1 client Module

This module provides some classes to simplify access to Odoo server via xmlrpc.

Example ussage of this module

```
>>> cl = Client('server.com', 'dbname', 'some_user', 'mypassword')
>>> sale_obj = cl['sale_order']
>>> sale_ids = sale_obj.search([('state','not in',['done','cancel'])])
>>> sale_data = sale_obj.read(sale_ids, ['name'])
>>> for order in sale_data:
...     print("%5s :    %s" % (order['id'],order['name']))
>>> product_tmpl_obj = cl['product.template']
>>> product_obj = cl['product.product']
>>> tmpl_ids = product_tmpl_obj.search([('name','ilike','template_name')])
>>> print(product_obj.search([('product_tmpl_id','in',tmpl_ids)]))

>>> db = Client('erp.host.com', 'dbname='db0', user='your_user')
>>> so = db['sale.order']
>>> order_ids = so.search([('state','=','done')])
>>> order = so.read(order_ids[0])
```

Also You can call any method (beside private ones starting with underscore(_)) of any model. For example following code allows to check availability of stock moves:

```
>>> db = session.connect()
>>> move_obj = db['stock.move']
>>> move_ids = [1234] # IDs of stock moves to be checked
>>> move_obj.check_assign(move_ids)
```

Ability to use Record class as analog to browse_record:

```
>>> move_obj = db['stock.move']
>>> move = move_obj.browse(1234)
>>> move.state
... 'confirmed'
>>> move.check_assign()
>>> move.refresh()
>>> move.state
... 'assigned'
>>> move.picking_id
... R('stock.picking', 12)['OUT-12']
>>> move.picking_id.id
... 12
>>> move.picking_id.name
... 'OUT-12'
>>> move.picking_id_.state
... 'assigned'
```

class odoo_rpc_client.client.**Client**(*host*, *dbname=None*, *user=None*, *pwd=None*, *port=8069*, *protocol='xml-rpc'*, *timeout=None*, ***extra_args*)

Bases: extend_me.Extensible

A simple class to connect to Odoo instance via RPC (XML-RPC, JSON-RPC) Should be initialized with following arguments:

Parameters

- **host** (*str*) – server host name to connect to
- **dbname** (*str*) – name of database to connect to
- **user** (*str*) – username to login as
- **pwd** (*str*) – password to log-in with
- **port** (*int*) – port number of server
- **protocol** (*str*) – protocol used to connect. To get list of available protcols call: odoo_rpc_client.connection.get_connector_names()
- **timeout** (*float*) – Connection timeout

any other keyword arguments will be directly passed to connector

Example:

```
>>> db = Client('host', 'dbname', 'user', pwd='Password')
>>> cl = Client('host')
>>> db2 = cl.login('dbname', 'user', 'password')
```

Allows access to Odoo objects / models via dictionary syntax:

```
>>> db['sale.order']
    Object ('sale.order')
```

clean_caches()
Clean client related caches

connect(***kwargs*)
Connects to the server

if any keyword arguments will be passed, new Proxy instnace will be created using folowing algorithm: get init args from self instance and update them with passed keyword arguments, and call Proxy class constructor passing result as arguments.

Note, that if You pass any keyword arguments, You also should pass 'pwd' keyword argument with user password

Returns Id of user logged in or new Client instance (if kwargs passed)

Return type intlClient

Raises ***LoginException*** – if wrong login or password

connection
Connection to server.

Return type *odoo_rpc_client.connection.connection.ConnectorBase*

database_version
Base database version ('8.0', '9.0', etc)

(Already parsed with `pkg_resources.parse_version`)

database_version_full
Full database base version ('9.0.1.3', etc)

(Already parsed with `pkg_resources.parse_version`)

dbname
Name of database to connect to

Return type str

execute(*obj*, *method*, **args*, ***kwargs*)
Call method *method* on object *obj* passing all next positional and keyword (if available on server) arguments to remote method

Note that passing keyword argments not available on OpenERp/Odoo server 6.0 and older

Parameters

- **obj** (*string*) – object name to call method for
- **method** (*string*) – name of method to call

Returns result of RPC method call

execute_wkf(*object_name*, *signal*, *object_id*)
Triggers workflow event on specified object

Parameters

- **object_name** (*string*) – send workflow signal for
- **signal** (*string*) – name of signal to send
- **object_id** – ID of document (record) to send signal to

classmethod from_url(*url*)
Create Client instance from URL

Parameters **url** (*str*) – url of Client

Returns Client instance

Return type *Client*

get_init_args()
Returns dictionary with init arguments which can be safely passed to class constructor

Return type dict

get_obj(*object_name*)
Returns wraper around Odoo object 'object_name' which is instance of orm.object.Object class

Parameters **object_name** – name of an object to get wraper for

Returns instance of Object which wraps choosen object

Return type *odoo_rpc_client.orm.object.Object*

get_url()
Returns dabase URL

At this moment mostly used internaly in session

host
Server host

Return type str

login(*dbname*, *user*, *password*)
Login to database

Return new Client instance. (Just an aliase on `connect` method)

Parameters

- **dbname** (*str*) – name of database to connect to
- **user** (*str*) – username to login as
- **password** (*str*) – password to log-in with

Returns new Client instance, with specifed credentials

Return type *odoo_rpc_client.client.Client*

plugins
Plugins associated with this Client instance

Return type *odoo_rpc_client.plugin.PluginManager*

Usage examples:

```
db.plugins.module_utils     # access module_utils plugin
db.plugins['module_utils]  # access module_utils plugin
```

port
Server port

protocol
Server protocol

Return type str

reconnect()
Recreates connection to the server and clears caches

Returns ID of user logged in

Return type int

Raises ***ClientException*** – if wrong login or password

ref (*xmlid*)
Return record for specified xmlid

Parameters **xmlid** (*str*) – string representing xmlid to get record for. xmlid must be *fully qualified* (with module name)

Returns Record for that xmlid or False

Return type *odoo_rpc_client.orm.record.Record*

registered_objects
List of registered in Odoo database objects

Return type list

server_version
Server base version ('8.0', '9.0', etc)

(Already parsed with `pkg_resources.parse_version`)

services
ServiceManager instance, which contains list of all available services for current connection.

Return type *odoo_rpc_client.service.service.ServiceManager*

Usage examples:

```
db.services.report    # report service
db.services.object    # object service (model related actions)
db.services.common    # used for login
                      # (db.services.common.login(dbname,
                      #                           username,
                      #                           password)
db.services.db        # database management service
```

classmethod to_url (*inst*, ***kwargs*)
Converts instance to url

Parameters **inst** (*Client* | *dict*) – instance to convert to init args

Returns generated URL

Return type str

uid
Returns ID of current user. if one is None, connects to database and returns it

Return type int

user
Currenct logged in user instance

Return type *odoo_rpc_client.orm.record.Record*

user_context
Get current user context

Return type dict

username
User login used to access DB

Return type str

2.1.2 exceptions Module

exception odoo_rpc_client.exceptions.**ClientException**
Bases: *odoo_rpc_client.exceptions.Error*

Base class for client related exceptions

exception odoo_rpc_client.exceptions.**ConnectorError**
Bases: *odoo_rpc_client.exceptions.Error*

Base class for exceptions related to connectors

exception odoo_rpc_client.exceptions.**Error**
Bases: Exception

Base class for exceptions

exception odoo_rpc_client.exceptions.**LoginException**
Bases: *odoo_rpc_client.exceptions.ClientException*

This exception should be raised, when operations requires login and password. For example interaction with Odoo object service.

exception odoo_rpc_client.exceptions.**ObjectException**
Bases: *odoo_rpc_client.exceptions.ClientException*

Base class for exceptions related to Objects

exception odoo_rpc_client.exceptions.**ReportError**
Bases: *odoo_rpc_client.exceptions.Error*

Error raise in process of report generation

2.1.3 plugin Module

class odoo_rpc_client.plugin.**Plugin**(*client*)
Bases: object

Base class for all plugins, extensible by name

(uses metaclass extend_me.ExtensibleByHashType)

Parameters **client** (*odoo_rpc_client.client.Client instance*) – instance of Client to bind plugins to

Example of simple plugin:

```
from odoo_rpc_client.plugin import Plugin

class AttandanceUtils(Plugin):

    # This is required to register Your plugin
    # *name* - is for db.plugins.<name>
    class Meta:
        name = "attendance"

    def get_sign_state(self):
        # Note: folowing code works on version 6 of Openerp/Odoo
        emp_obj = self.client['hr.employee']
        emp_id = emp_obj.search(
            [('user_id', '=', self.client.uid)])
```

(continues on next page)

(continued from previous page)

```
        emp = emp_obj.read(emp_id, ['state'])
        return emp[0]['state']
```

This plugin will automaticaly register itself in system, when module which contains it will be imported.

client
Related Client instance

class odoo_rpc_client.plugin.**PluginManager**(*client*)
Bases: extend_me.Extensible, *odoo_rpc_client.utils.DirMixIn*

Class that holds information about all plugins

Parameters **client** (*odoo_rpc_client.client.Client instance*) – instance of Client to bind plugins to

Plugiins will be accessible via index or attribute syntax:

```
plugins = PluginManager(client)
plugins.Test   # acceps plugin 'Test' as attribute
plugins['Test']  # access plugin 'Test' via indexing
```

refresh()
Clean-up plugin cache This will force to reinitialize each plugin when asked

registered_plugins
List of names of registered plugins

class odoo_rpc_client.plugin.**TestPlugin**(*client*)
Bases: *odoo_rpc_client.plugin.Plugin*

Jusn an example plugin to test if plugin logic works

class Meta
Bases: object

name = 'Test'

test()

2.1.4 utils Module

class odoo_rpc_client.utils.**AttrDict**
Bases: dict, *odoo_rpc_client.utils.DirMixIn*

Simple class to make dictionary able to use attribute get operation to get elements it contains using syntax like:

```
>>> d = AttrDict(arg1=1, arg2='hello')
>>> print(d.arg1)
    1
>>> print(d.arg2)
    hello
>>> print(d['arg2'])
    hello
>>> print(d['arg1'])
    1
```

class odoo_rpc_client.utils.**DirMixIn**
Bases: object

class odoo_rpc_client.utils.**UConverter**(*hint_encodings=None*)
Bases: object

Simple converter to unicode

Create instance with specified list of encodings to be used to try to convert value to unicode

Example:

```
ustr = UConverter(['utf-8', 'cp-1251'])
my_unicode_str = ustr(b'hello - ')
```

default_encodings = ['utf-8', 'ascii']

odoo_rpc_client.utils.**wpartial**(*func*, **args*, ***kwargs*)
Wrapped partial, same as functools.partial decorator, but also calls functools.wrap on its result thus shwing correct function name and representation.

2.1.5 Subpackages

connection Package

connection Module

odoo_rpc_client.connection.connection.**get_connector**(*name*)
Return connector specified by it's name

odoo_rpc_client.connection.connection.**get_connector_names**()
Returns list of connector names registered in system

class odoo_rpc_client.connection.connection.**ConnectorBase**(*host*, *port*, *timeout=None*, *extra_args=None*)
Bases: object

Base class for all connectors

Parameters

- **host** (*str*) – hostname to connect to
- **port** (*int*) – port to connect to
- **extra_args** (*dict*) – extra arguments for specific connector.

extra_args
Connector extra arguments

get_service(*name*)
Returns service for specified *name*

Parameters **name** – name of service

Returns specified service instance

host
Connector host

port
Connector port

timeout
Connector timeout

update_extra_args(***kwargs*)
Update extra args and clean service cache

jsonrpc Module

class odoo_rpc_client.connection.jsonrpc.**ConnectorJSONRPC**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.connection.connection.ConnectorBase*

JSON-RPC connector

available extra arguments:

- ssl_verify: (optional) if True, the SSL cert will be verified.

class Meta
Bases: object

name = 'json-rpc'

use_ssl = False

class odoo_rpc_client.connection.jsonrpc.**ConnectorJSONRPCS**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.connection.jsonrpc.ConnectorJSONRPC*

JSON-RPCS Connector

class Meta
Bases: object

name = 'json-rpcs'

use_ssl = True

exception odoo_rpc_client.connection.jsonrpc.**JSONRPCError**(*message*, *code=None*, *data=None*)
Bases: *odoo_rpc_client.exceptions.ConnectorError*

JSON-RPC error wrapper

data_debug
Debug information got from Odoo server

Usualy traceback

data_message
Error message got from Odoo server

class odoo_rpc_client.connection.jsonrpc.**JSONRPCMethod**(*rpc_proxy*, *url*, *service*, *method*)
Bases: object

Class that implements RPC call via json-rpc protocol

prepare_method_data(**args*)
Prepare data for JSON request

class odoo_rpc_client.connection.jsonrpc.**JSONRPCProxy**(*host*, *port*, *service*, *ssl=False*, *ssl_verify=True*, *timeout=None*)
Bases: object

Simple Odoo service proxy wrapper

xmlrpc Module

class odoo_rpc_client.connection.xmlrpc.**ConnectorXMLRPC**(*host*, *port*, *timeout=None*, *extra_args=None*)

Bases: *odoo_rpc_client.connection.connection.ConnectorBase*

XML-RPC connector

Note: extra_arguments may be same as parametrs of xmlrpclib.ServerProxy

class Meta

Bases: object

name = 'xml-rpc'

ssl = False

get_service_url(*service_name*)

class odoo_rpc_client.connection.xmlrpc.**ConnectorXMLRPCS**(*host*, *port*, *timeout=None*, *extra_args=None*)

Bases: *odoo_rpc_client.connection.xmlrpc.ConnectorXMLRPC*

XML-RPCS Connector

Note: extra_arguments may be same as parametrs of xmlrpclib.ServerProxy

class Meta

Bases: object

name = 'xml-rpcs'

ssl = True

exception odoo_rpc_client.connection.xmlrpc.**XMLRPCError**(*fault_instance*)

Bases: *odoo_rpc_client.exceptions.ConnectorError*

Exception raised on XMLRpc errors

Parameters **fault_instance** (*xmlrpclib.Fault*) – exception raised by XMLRPC lib

fault

Return xmlrpclib.Fault instance related to this error

class odoo_rpc_client.connection.xmlrpc.**XMLRPCMethod**(*method*)

Bases: object

Class wrapper around XML-RPC method to wrap xmlrpclib.Fault into XMLRPCProxy

class odoo_rpc_client.connection.xmlrpc.**XMLRPCProxy**(*uri*, *timeout=None*, **args*, ***kwargs*)

Bases: xmlrpc.client.ServerProxy

Wrapper class around XML-RPC's ServerProxy to wrap method's errors into XMLRPCError class

service Package

odoo_rpc_client.service.**get_service_class**(*name*)

Return service class specified by it's name

class odoo_rpc_client.service.**ServiceBase**(*service*, *client*, *name*)

Bases: *object*

Base class for all Services

Parameters

- **service** – instance of original service class. must support folowing syntax `service.service_method(args)` to call remote methods
- **client** – instance of Client, this service is binded to

clean_cache()
To be implemented by subclasses, if needed

client
Related Client instance

name
Service name

class odoo_rpc_client.service.**ServiceManager**(*client*)
Bases: `extend_me.Extensible`, *odoo_rpc_client.utils.DirMixIn*

Class to hold services related to specific client and to automaticaly clean service cached on update of service classes

Usage:

```
services = ServiceManager(client)
services.service_list          # get list of registered services
services.object                # returns service with name 'object'
services['common']             # returns service with name 'common'
services.get_service('report') # returns service named 'report'
```

clean_cache()
Cleans manager's service cache.

classmethod clean_caches()
Cleans saved service instances, so on next access new service instances will be generated. This usualy happens when new service extension enabled (new class inherited from ServiceBase created)

clean_service_caches()
Clean caches of all services handled by this mananger usualy this should be called on module update, when list of available objects or reports changed

client
Client instance this ServiceManager is bounded to

get_service(*name*)
Returns instance of service with specified name

Parameters **name** – name of service

Returns specified service instance

service_list
Returns list of all registered services

db Module

class odoo_rpc_client.service.db.**DBService**(*service*, *client*, *name*)
Bases: *odoo_rpc_client.service.service.ServiceBase*

Service class to simplify interaction with 'db' service

class Meta
Bases: object

name = 'db'

create_db(*password*, *dbname*, *demo=False*, *lang='en_US'*, *admin_password='admin'*)
Create new database on server, named *dbname*

Parameters

- **password** (*str*) – super admin password
- **dbname** (*str*) – name of database to create
- **demo** (*bool*) – load demo data or not. Default: False
- **lang** (*str*) – language to be used for database. Default: 'en_US'
- **admin_password** (*str*) – password to be used for 'Administrator' database user. Default: 'admin'

Returns Client instance logged to created database as admin user.

Return type instance of *odoo_rpc_client.client.Client*

db_exist(*db*)
Check if database exists

Parameters **db** (*str|Client*) – name of database or *Client* instance with *client.dbname is not None*

Returns True if database exists else False

Return type bool

drop_db(*password*, *db*)
Drop specified database

Parameters

- **password** (*str*) – super admin password
- **db** (*str|Client*) – name of database or *Client* instance with *client.dbname is not None*

Raise *ValueError* (unsupported value of *db* argument)

dump_db(*password*, *db*, ***kwargs*)
Dump database

Note, that from defined arguments, may be passed other arguments (for example odoo version 9.0 requires format arg to be passed)

Note, this method may consume huge amout of memory. In production dump/restore have to be done by other means.

Parameters

- **password** (*str*) – super admin password
- **db** (*str|Client*) – name of database or *Client* instance with *client.dbname is not None*
- **format** (*str*) – (only odoo 9.0) (default: zip)

Raise *ValueError* (unsupported value of *db* argument)

Returns byte-string with base64 encoded data

Return type bytes

list_db()
Display list of databses of thist connection

restore_db(*password*, *dbname*, *data*, ***kwargs*)
Restore database

Note, this method may consume huge amout of memory. In production dump/restore have to be done by other means.

Parameters

- **password** (*str*) – super admin password
- **dbname** (*str*) – name of database
- **data** (*bytes*) – restore data (base64 encoded string)
- **copy** (*bool*) – (only odoo 8.0+) if set to True, then new db-uid will be generated. (default: False)

Returns True

Return type bool

server_base_version()
Returns server base version ('9.0', '8.0', etc) parsed via pkg_resources.parse_version. No info about comunity / enterprise here

server_version()
Returns server version.

(Already parsed with pkg_resources.parse_version)

server_version_str()
Return server version (not wrapped by pkg.parse_version)

object Module

class odoo_rpc_client.service.object.**ObjectService**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.service.service.ServiceBase*

Service class to simplify interaction with 'object' service Particulary, implements logic of choosing execute method ('execute' or 'execute_kw') The last one cannot work with keyword arguments(

class Meta
Bases: object

name = 'object'

clean_cache()
Cleans service cache, to fill them with fresh data on next call of related methods

execute(*obj*, *method*, **args*, ***kwargs*)
First arguments should be 'object' and 'method' and next will be passed to method of given object

execute_wkf(*object_name*, *signal*, *object_id*)
Triggers workflow event on specified object

Parameters

- **object_name** (*str*) – name of object/model to trigger workflow on
- **signal** (*str*) – name of signal to send to workflow

- **object_id** (*int*) – ID of document (record) to send signal to

get_registered_objects()
Returns list of registered objects in database

report Module

Report printing logic

Best way to generate report is:

```
data_records = client['res.partner'].search_records([], limit=10)
report = client.services.report['res.partner'].generate(data_records)
report.content
```

Or if it is desired to save it on disk:

```
data_records = client['res.partner'].search_records([], limit=10)
report = client.services.report['res.partner'].generate(data_records)
report.save('filename to save report with')
```

where *report* is instance of *ReportResult* and *report.content* returns already *base64* decoded content of report, which could be directly written to file (or just use *report.save(path)* method)

class odoo_rpc_client.service.report.**Report**(*service*, *report*)
Bases: extend_me.Extensible

Class that represents report.

useful to simplify report generation

Parameters

- **service** (ReportService) – instance of report service to bind report to
- **report** (Record) – model of report action

generate(*model_data*, *report_type='pdf'*, *context=None*)
Generate report

Parameters

- **model_data** – RecordList or Record or list of obj_ids. represent document or documents to generate report for
- **report_type** (*str*) – Type of report to generate. default is 'pdf'.
- **context** (*dict*) – Aditional info. Optional.

Raises ReportError

Returns ReportResult instance that contains generated report

Return type *ReportResult*

name
Name of report

report_action
Action of this report

service
Service this report is binded to

class odoo_rpc_client.service.report.**ReportResult**(*report*, *result*, *path=None*)
Bases: extend_me.Extensible

Just a simple and extensible wrapper on report result

As variant of usage - wrap result returned by server methods `report_get` and `render_report` like:

```
ReportResult(report_get(report_id))
```

content
Report file content. Already base64-decoded

format
Report format

path
Path where file is located or will be located on save

result
Base64-encoded report content. To get already decoded report content, use `.content` property

Raises ***ReportError*** – When .state property is False. This may appear in case when report is not ready yet, when using *report* and *report_get* methods

save(*path=None*)
Save's file by specified path or if no path specified save it in temp dir with automaticly generated name.

state
Result status. only if True, other fields are available

class odoo_rpc_client.service.report.**ReportService**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.service.service.ServiceBase*

Service class to simplify interaction with 'report' service

class Meta
Bases: object

name = 'report'

available_reports
Returns dictionary with all available reports

{<report name> : <Report instance>}

generate_report(*report_name*, *report_data*, *report_type='pdf'*, *context=None*)
Generate specified report for specifed report data. Report data could be RecordList or Record instance. Result is wrapped into ReportResult class

Parameters

- **report_name** (*str*) – string representing name of report service
- **report_data** – RecordList or Record or ('model_name', obj_ids) represent document or documents to generate report for
- **report_type** (*str*) – Type of report to generate. default is 'pdf'.
- **context** (*dict*) – Aditional info. Optional.

Raises ReportError

Returns ReportResult instance that contains generated report

Return type *ReportResult*

render_report(*report_name*, *model*, *ids*, *report_type='pdf'*, *context=None*)
Proxy to report service *render_report* method

NOTE: available after version 6.1.

Parameters

- **report_name** (*str*) – string representing name of report service
- **model** (*str*) – name of model to generate report for
- **ids** (*list of int | int*) – list of object ID to get report for (or just single id)
- **report_type** (*str*) – Type of report to generate. default is 'pdf'.
- **context** (*dict*) – Aditional info. Optional.

Returns dictinary with keys: - 'state': boolean, True if report generated correctly - 'result': base64 encoded content of report file - 'format': string representing report format

Return type dict

report(*report_name*, *model*, *ids*, *report_type='pdf'*, *context=None*)
Proxy to report service *report* method

Parameters

- **report_name** (*str*) – string representing name of report service
- **model** (*str*) – name of model to generate report for
- **ids** (*list of int | int*) – list of object ID to get report for (or just single id)
- **report_type** (*str*) – Type of report to generate. default is 'pdf'.
- **context** (*dict*) – Aditional info. Optional.

Returns ID of report to get by method *report_get*

Return type int

report_get(*report_id*)
Proxy method to report service *report_get* method

Parameters **report_id** (*int*) – int that represents ID of report to get (value returned by report method)

Returns

dictinary with keys:

- 'state': boolean, True if report generated correctly
- 'result': base64 encoded content of report file
- 'format': string representing format, report generated in

Return type dict

service Module

odoo_rpc_client.service.service.**get_service_class**(*name*)
Return service class specified by it's name

class odoo_rpc_client.service.service.**ServiceBase**(*service*, *client*, *name*)
Bases: object

Base class for all Services

Parameters

- **service** – instance of original service class. must support folowing syntax `service.service_method(args)` to call remote methods
- **client** – instance of Client, this service is binded to

clean_cache()
To be implemented by subclasses, if needed

client
Related Client instance

name
Service name

class odoo_rpc_client.service.service.**ServiceManager**(*client*)
Bases: extend_me.Extensible, *odoo_rpc_client.utils.DirMixIn*

Class to hold services related to specific client and to automaticaly clean service cached on update of service classes

Usage:

```
services = ServiceManager(client)
services.service_list          # get list of registered services
services.object                # returns service with name 'object'
services['common']             # returns service with name 'common'
services.get_service('report') # returns service named 'report'
```

clean_cache()
Cleans manager's service cache.

classmethod clean_caches()
Cleans saved service instances, so on next access new service instances will be generated. This usualy happens when new service extension enabled (new class inherited from ServiceBase created)

clean_service_caches()
Clean caches of all services handled by this mananger usualy this should be called on module update, when list of available objects or reports changed

client
Client instance this ServiceManager is bounded to

get_service(*name*)
Returns instance of service with specified name

Parameters **name** – name of service

Returns specified service instance

service_list
Returns list of all registered services

orm Package

object Module

class odoo_rpc_client.orm.object.**Object**(*service*, *object_name*)

Bases: *odoo_rpc_client.utils.DirMixIn*

Base class for all Objects

Provides simple interface to remote osv.osv objects:

```
erp = Client(...)
sale_obj = Object(erp, 'sale.order')
sale_obj.search([('state','not in',['done','cancel'])])
```

To create new instance - use *get_object* function, it implements all extensions magic, whic is highly used in this project

It is posible to create extension only to specific object. Example could be found in `plugins/module_utils.py` file.

client

Client instance, this object is relatedto

Return type *odoo_rpc_client.client.Client*

columns_info

Reads information about fields available on model.

Internaly this method uses *fields_get* method.

Returns dictionary with information about fields available on this model.

Return type *odoo_rpc_client.utils.AttrDict*

create(*vals*, *context=None*)

Create new record with *vals*

Also look at Odoo documentation for this method

Parameters

- **vals** (*dict*) – dictionary with values to be written to newly created record
- **context** (*dict*) – context dictionary

Returns ID of newly created record

Return type int

name

Name of the object

Return type str

read(*ids*, *fields=None*, *context=None*)

Read *fields* for records with id in *ids*

Also look at Odoo documentation for this method

Parameters

- **ids** (*int | list*) – ID or list of IDs of records to read data for
- **fields** (*list*) – list of field names to read. if not passed all fields will be read.
- **context** (*dict*) – dictionary with extra context

Returns list of dictionaries with data had been read

Return type list

resolve_field_path(*field*)
Resolves dot-separated field path to list of tuples (model, field_name, related_model)

Parameters **field** (*str*) – dot-separated field path to resolve

For example:

```
sale_obj = client['sale.order']
sale_obj.resolve_field_path('partner_id.country_id.name')
```

will be resoved to:

```
[('sale.order', 'partner_id', 'res.partner'),
 ('res.partner', 'country_id', 'res.country'),
 ('res.country', 'name', False)]
```

search(*args[, offset=0][, limit=None][, order=None][, count=False][, context=None]*)
Search records by criteria.

Also look at Odoo documentation for this method

search_count(*domain=None*, *context=None*)
Returns the number of records matching the provided domain.

Returns number of recods

Return type int

search_read(*domain=None*, *fields=None*, *offset=0*, *limit=None*, *order=None*, *context=None*)
Search and read records specified by domain

Note that this method reads data in correct order

Also look at Odoo documentation

Returns list of dictionaries with data had been read

Return type list

service
Object service instance

stdcall_methods
Property that returns all methods of this object, that supports standard call

Returns list with names of *stdcall* methods

Return type list(str)

unlink(*ids*, *context=None*)
Unlink records specified by *ids*

Also look at Odoo documentation for this method

Parameters **ids** (*list*) – list of IDs of records to be deleted

write(*ids*, *vals*, *context=None*)
Write data in *vals* dictionary to records with ID in *ids*

For more info, look at odoo documentation for this method

Parameters

- **ids** (*int | list*) – ID or list of IDs of records to write data for
- **vals** (*dict*) – dictinary with values to be written to database for records specified by ids
- **context** (*dict*) – context dictionary

odoo_rpc_client.orm.object.**get_object**(*client*, *name*)

Create new Object instance.

Parameters

- **client** (Client) – Client instance to bind this object to
- **name** (*str*) – name of object. Ex. 'sale.order'

Returns Created Object instance

Return type *Object*

cache Module

odoo_rpc_client.orm.cache.**empty_cache**(*client*)

Create instance of empty cache for Record

Parameters **client** (Client) – instance of Client to create cache for

Returns instance of Cache class

Return type *Cache*

Cache is dictionary-like object with structure like:

```
cache = {
    'product.product': {
        1: {
            'id': 1,
            'name': 'product1',
            'default_code': 'product1',
        },
    },
}
```

class odoo_rpc_client.orm.cache.**Cache**(*client*, **args*, ***kwargs*)

Bases: dict

Cache to be used for Record's data.

This is root cache, which manages model local cache

cache['res.partner'] -> ObjectCache('res.partner')

client

Access to Client instance this cache belongs to

class odoo_rpc_client.orm.cache.**ObjectCache**(*root*, *obj*, **args*, ***kwargs*)

Bases: dict

Cache for object / model data

Automatically generates empty data dicts for records requested. Also contains object context

cache_field(*rid*, *ftype*, *field_name*, *value*)

This method impelment additional caching functionality, like caching related fields, and so...

Parameters

- **rid** (*int*) – Record ID
- **ftype** (*str*) – field type
- **field_name** (*str*) – name of field
- **value** – value to cache for field

context
Return context instance related to this cache

get_ids_to_read(**fields*)
Return list of ids, that have no at least one of specified fields in cache

For example:

```
cache.get_ids_to_read('name', 'country_id', 'parent_id')
```

This code will traverse all record ids managed by this cache, and find those that have no at least one field in cache. This is highly useful in prefetching

parse_prefetch_fields(*fields*)
Parse fields to be prefetched, sparating, cache's object fields and related fields.

Used internaly

Parameters **fields** (*list*) – list of fields to prefetch

Returns returns `tuple(prefetch_fields, related_fields)`, where `prefetch_fields` is list of fields, to be read for current object, and `related_fields` is dictionary of form: `{'related.object': ['relatedfield1', 'relatedfield2.relatedfield']}`

Return type tuple

prefetch_fields(*fields*)
Prefetch specified fields for this cache. Also, dot (".") may be used in field name to prefetch related fields:

```
cache.prefetch_fields(
    ['myfield1', 'myfields2_ids.relatedfield'])
```

Parameters **fields** (*list*) – list of fields to prefetch

update_context(*new_context*)
Updates or sets new context for thes ObjectCache instance

Parameters **new_context** (*dict*) – context dictionary to update cached context with

Returns updated context

update_keys(*keys*)
Add new IDs to cache.

Parameters **keys** (*list*) – list of new IDs to be added to cache

Returns self

Return type *ObjectCache*

record Module

This module contains classes and logic to handle operations on records

class odoo_rpc_client.orm.record.**Record**(*obj*, *rid*, *cache=None*, *context=None*)

Bases: *odoo_rpc_client.utils.DirMixIn*

Base class for all Records

Do not use it to create record instances manualy. Use `get_record` function instead. It implements all extensions mangic

But class should be used for `isinstance` checks.

It is posible to create extensions of this class that will be binded only to specific Odoo objects

For example, if You need to extend all recrods of products, do something like this:

```
class MyProductRecord(Record):
    class Meta:
        object_name = 'product.product'

    def __init__(self, *args, **kwargs):
        super(MyProductRecord, self).__init__(*args, **kwargs)

        # to avoid double read, save once read value to record
        # instance
        self._sale_orders = None

    @property
    def sale_orders(self):
        ''' Sale orders related to curent product
        '''
        if self._sale_orders is None:
            so = self._client['sale.order']
            domain = [('order_line.product_id', '=', self.id)]
            self._sale_orders = so.search_records(
                                    domain, cache=self._cache)
        return self._sale_orders
```

And atfter this, next code is valid:

```
products = client['product.product'].search_records([])
products_so = products.filter(lambda p: bool(p.sale_orders))
products_so_gt_10 = products.filter(
    lambda p: len(p.sale_orders) > 10)

for product in products_so_gt_10:
    print("Product: %s" % product.default_code)
    for pso in product.sale_orders:
        print("     %s" % pso.name)
```

Parameters

- **obj** (Object) – instance of object this record is related to
- **rid** (*int*) – ID of database record to fetch data from
- **cache** (Cache) – Cache instance. (usualy generated by function empty_cache())
- **context** (*dict*) – if specified, then cache's context will be updated

Note, to create instance of cache call *empty_cache*

as_dict
Provides dictionary with record's data in raw form

Return type dict

context
Returns context to be used for thist record

copy(*default=None*, *context=None*)
copy this record.

Parameters

- **default** (*dict*) – dictionary default values for new record (optional)
- **context** (*dict*) – dictionary with context used to copy this record. (optional)

Returns Record instance for created record

Return type *Record*

Note about context: by default cache's context will be used, and if some context will be passed to this method, new dict, which is combination of default context and passed context, will be passed to server.

get(*field_name*, *default=None*)
Try to get field *field_name*, if if field name is not available return *default* value for it

if *default* is None and it is not possible to get field value, then raises *KeyErro*

Parameters

- **field_name** (*str*) – name of field to get value for
- **default** – default value for case when no such field

Returns field value

Raises **KeyError** – if cannot get field value

Note: This may be useful for code that expected to be working for different Odoo versions which have different database schemes.

id
Record ID

Return type int

read(*fields=None*, *context=None*, *multi=False*)
Rereads data for this record (or for al records in whole cache)

Parameters

- **fields** (*list*) – list of fields to be read (optional)
- **context** (*dict*) – context to be passed to read (optional) does not midify record's context
- **multi** (*bool*) – if set to True, that data will be read for all records of this object in current cache (query).

Returns dict with data had been read

Return type dict

refresh()
Reread data and clean-up the caches

Returns self

Return type *Record*

class odoo_rpc_client.orm.record.**ObjectRecords**(**args*, ***kwargs*)

Bases: *odoo_rpc_client.orm.object.Object*

Adds support to use records from Object classes

browse(**args*, ***kwargs*)

Aliase to *read_records* method. In most cases same as serverside *browse* (i mean server version 7.0)

create_record(*vals*, *context=None*, *cache=None*)

Create new record in database and return Record instance. Same as *create* method, but returns Record instance instead of ID.

Parameters

- **vals** (*dict*) – values to create record with
- **context** (*dict*) – extra context to pass to *create* method
- **cache** (Cache) – cache to add created record to. if None is passed, then new cache will be created.

Returns Record instance of created record

Return type *odoo_rpc_client.orm.record.Record*

For example:

```
>>> partner_obj = db['res.partner']
>>> john = partner_obj.create_record({'name': 'John'})
>>> john.name
John
```

model

Returns Record instance of model related to this object. Useful to get additional info on object.

Returns Record('ir.model')

Return type *odoo_rpc_client.orm.record.Record*

model_name

Result of name_get called on object's model

read_records(*ids*, *fields=None*, *context=None*, *cache=None*)

Return instance or RecordList class, making available to work with data simpler

Parameters

- **ids** (*int|list of int*) – ID or list of IDS to read data for
- **fields** (*list*) – list of fields to read (*optional*)
- **context** (*dict*) – context to be passed to read. default=None
- **cache** (Cache) – cache to use for records and record lists. Pass None to create new cache. default=None.

Returns Record instance if *ids* is int or RecordList instance if *ids* is list of ints

Return type Record|RecordList

For example:

```
>>> so_obj = db['sale.order']
>>> data = so_obj.read_records([1,2,3,4,5])
>>> for order in data:
        order.write({'note': 'order data is %s'%order.data})
```

search_records(**args*, ***kwargs*)

Return instance or list of instances of Record class, making available to work with data simpler

Parameters

- **domain** – list of tuples, specifying search domain
- **offset** (*int*) – (optional) number of results to skip in the returned values (default:0)
- **limit** (*int|False*) – optional max number of records in result (default: False)
- **order** (*str*) – optional columns to sort
- **context** (*dict*) – optional context to pass to *search* method
- **count** – if set to True, then only amount of recrods found will be returned. (default: False)
- **read_fields** (*list of strings*) – optional. specifies list of fields to read.
- **cache** (Cache) – cache to be used for records and recordlists

Returns RecordList contains records found, or integer that represents amount of records found (if count=True)

Return type RecordListlint

For example:

```
>>> so_obj = db['sale.order']
>>> data = so_obj.search_records([('date','>=','2013-01-01')])
>>> for order in data:
...     order.write({'note': 'order date is %s'%order.date})
```

simple_fields

List of simple fields which could be fetched fast enough

This list contains all fields that are not function nor binary

Type list of strings

class odoo_rpc_client.orm.record.**RecordList**(*obj*, *ids=None*, *fields=None*, *cache=None*, *context=None*)

Bases: collections.abc.MutableSequence, *odoo_rpc_client.utils.DirMixIn*

Class to hold list of records with some extra functionality

Parameters

- **obj** (Object) – instance of Object to make this list related to
- **ids** (*list of int*) – list of IDs of objects to read data from
- **fields** (*list of strings*) – list of field names to read by default
- **cache** (Cache) – Cache instance. (usualy generated by function empty_cache()
- **context** (*dict*) – context to be passed automatically to methods called from this list (not used yet)

context
Returns context to be used for this list

copy(*context=None*, *new_cache=False*)
Returns copy of this list, possibly with modified context and new empty cache.

Parameters

- **context** (*dict*) – new context values to be used on new list
- **new_cache** (*bool*) – if set to True, then new cache instance will be created for resulting recordlist if set to Cache instance, than it will be used for resulting recordlist

Returns copy of this record list.

Return type *RecordList*

Raises **ValueError** – when incorrect value passed to new_cache

existing(*uniqify=True*)
Filters this list with only existing items

Parm bool uniqify if set to True, then all dublicates will be removed. Default: True

Returns new RecordList instance

Return type *RecordList*

filter(*func*)
Filters items using *func*.

Parameters **func** (*callable(record)->bool|anyfield.SField*) – callable to check if record should be included in result.

Returns RecordList which contains records that matches results

Return type *RecordList*

group_by(*grouper*)
Groups all records in list by specifed grouper.

Parameters **grouper** (*string|callable(record)|anyfield.SField*) – field name or callable to group results by. if callable is passed, it should receive only one argument - record instance, and result of calling grouper will be used as key to group records by.

Returns dictionary

for example we have list of sale orders and want to group it by state

```
# so_list - variable that contains list of sale orders selected
# by some criterias. so to group it by state we will do:
group = so_list.group_by('state')

# Iterate over resulting dictionary
for state, rlist in group.iteritems():
    # Print state and amount of items with such state
    print state, rlist.length
```

or imagine that we would like to group records by last letter of sale order number

```
# so_list - variable that contains list of sale orders selected
# by some criterias. so to group it by last letter of sale
# order name  we will do:
```

(continues on next page)

(continued from previous page)

```
group = so_list.group_by(lambda so: so.name[-1])

# Iterate over resulting dictionary
for letter, rlist in group.iteritems():
    # Print state and amount of items with such state
    print letter, rlist.length
```

ids
IDs of records present in this RecordList

insert (*index*, *item*)
Insert record to list

Parameters

- **item** (*Record|int*) – Record instance to be inserted into list. if int passed, it considered to be ID of record
- **index** (*int*) – position where to place new element

Returns self

Return type *RecordList*

length
Returns length of this record list

mapped (*field*)
Experimental, Provides similar functionality to Odoo's mapped() method, but supports only dot-separated field name as argument, no callables yet.

Returns list of values of field of each record in this recordlist. If value of field is RecordList or Record instance, than RecordList instance will be returned

Thus folowing code will work

```
# returns a list of names
records.mapped('name')

# returns a recordset of partners
record.mapped('partner_id')

# returns the union of all partner banks,
# with duplicates removed
record.mapped('partner_id.bank_ids')
```

Parameters **field** (*str*) – returns list of values of 'field' for each record in this RecordList

Return type list or *RecordList*

object
Object this record is related to

prefetch (**fields*)
Prefetches specified fields into cache if no fields passed, then all 'simple_fields' will be prefetched

By default field read performed only when that field is requested, thus when You need to read more then one field, few rpc requests will be performed. to avoid multiple unneccessary rpc calls this method is implemented.

Returns self, which allows chaining of operations

Return type *RecordList*

read(*fields=None*, *context=None*)
Read wrapper. Takes care about adding RecordList's context to object's read method.

Warning: does not update cache by data been read

records
Returns list (class 'list') of records

refresh()
Cleanup data caches. next try to get data will cause rereading of it

Returns self

Return type instance of RecordList

search(*domain*, **args*, ***kwargs*)
Performs normal search, but adds `('id', 'in', self.ids)` to search domain

Returns list of IDs found

Return type list of integers

search_records(*domain*, **args*, ***kwargs*)
Performs normal search_records, but adds `('id', 'in', self.ids)` to domain

Returns RecordList of records found

Return type RecordList instance

sort(*key=None*, *reverse=False*)
sort(key=None, reverse=False) – inplace sort

anyfield.SField instances may be safely passed as 'key' arguments. no need to convert them to function explicitly

Returns self

odoo_rpc_client.orm.record.**get_record**(*obj*, *rid*, *cache=None*, *context=None*)
Creates new Record instance

Use this method to create new records, because of standard object creation bypasses extension's magic.

param Object obj instance of Object this record is related to

param int rid ID of database record to fetch data from

param cache Cache instance. (usualy generated by function empty_cache()

type cache Cache

param dict context if specified, then cache's context will be updated

return created Record instance

rtype Record

odoo_rpc_client.orm.record.**get_record_list**(*obj*, *ids=None*, *fields=None*, *cache=None*, *context=None*)
Returns new instance of RecordList object.

Parameters

- **obj** (*Object*) – instance of Object to make this list related to
- **ids** (*list of int*) – list of IDs of objects to read data from

- **fields** (*list of strings (not used now)*) – list of field names to read by default (not used now)
- **cache** (Cache) – Cache instance. (usualy generated by function empty_cache()
- **context** (*dict*) – context to be passed automatically to methods called from this list (not used yet)

service Module

class odoo_rpc_client.orm.service.**Service**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.service.object.ObjectService*

Service class to simplify interaction with 'object' service. Particulary, implements logic of choosing execute method ('execute' or 'execute_kw') to use. The last one cannot work with keyword arguments

clean_cache()
Cleans caches, to fill them with fresh data on next call of related methods

get_obj(*object_name*)
Returns wraper around Odoo object 'object_name' which is instance of Object

Parameters **object_name** (*string*) – name of an object to get wraper for

Returns instance of Object which wraps choosen object

Return type *Object*

2.2 Plugins Package

This package contains plugins provided out-of-the-box

2.2.1 module_utils Plugin

class odoo_rpc_client.plugins.module_utils.**ModuleObject**(*service*, *object_name*)
Bases: *odoo_rpc_client.orm.object.Object*

Add shortcut methods to 'ir.module.module' object / model to install or upgrade modules

Also this methods will be available for Record instances too

class Meta
Bases: object

name = 'ir.module.module'

install(*ids*, *context=None*)
Immediatly install module

upgrade(*ids*, *context=None*)
Immediatly upgrades module

class odoo_rpc_client.plugins.module_utils.**ModuleUtils**(**args*, ***kwargs*)
Bases: *odoo_rpc_client.plugin.Plugin*, *odoo_rpc_client.utils.DirMixIn*

Utility plugin to simplify module management

Allows to access Odoo module objects as attributes of this plugin:

```
# this method supports IPython autocomplete
db.plugins.module_utils.m_stock
```

or dictionary style access to modules:

```
db.plugins.moduld_utils['stock']
```

which is equivalent to

```
db.get_obj('ir.module.module').search_records(
    [('name','=','stock')])[0]
```

Also autocomplete in IPython supported for this syntax

class Meta
Bases: object

name = 'module_utils'

installed_modules
RecordList with list of modules installed in currenct database

Return type *RecordList*

modules
Returns dictionary of modules registered in system.

Result dict is like: `{'module_name': module_inst}`

where *module_inst* is *Record* instance for this module

update_module_list()
Update module list

If there are some modules added to server, update list, to be able to installe them.

2.2.2 external_ids Plugin

class odoo_rpc_client.plugins.external_ids.**ExternalIDS**(*client*)
Bases: *odoo_rpc_client.plugin.Plugin*

This plugin adds aditional methods to work with external_ids (xml_id) for Odoo records.

class Meta
Bases: object

name = 'external_ids'

get_for(*val*, *module=None*)
Return RecordList of 'ir.model.data' for val or False

Parameters

- **val** – value to get 'ir.model.data' records for
- **module** (*str*) – module name to search 'ir.model.data' for

Return type *RecordList*

Returns RecordList with 'ir.model.data' records found

Raises **ValueError** – if *val* argument could not be parsed

`val` could be one of folowing types:

- `Record` instance
- `RecordList` instance
- `tuple(model, res_id)`, for example `('res.partner', 5)`
- `str`, string in format 'module.name'.

Note, in case of *val* is *str*: if *module* specified as parameter, then *val* supposed to be *name* only. For example, folowing calls are equal:

```
cl.plugins.external_ids.get_for('base.group_configuration')
cl.plugins.external_ids.get_for('group_configuration',
                                module='base')
```

get_record(*xml_id*, *module=None*)

Return *Record* instance for specified xml_id

Parameters

- **xml_id** (*str*) – string with xml_id to search record for
- **module** (*str*) – module name to search Record in

Return type *Record*

Returns Record for *val* or False if not found

Raises **ValueError** – if *xml_id* argument could not be parsed

get_xmlid(*val*, *module=None*)

Return *xml_id* for *val*. Note, that only first xml_id will be returned!

Parameters

- **val** – look in documentation for *get_for* method
- **module** (*str*) – module name to search xml_id for

Return type str

Returns xml_id for *val* or False if not found

Raises **ValueError** – if *val* argument could not be parsed

Note, that if *module* specified as parametr, then *val* supposed to be *name* only

class odoo_rpc_client.plugins.external_ids.**Record__XMLIDS**(*obj*, *rid*, *cache=None*, *context=None*)

Bases: *odoo_rpc_client.orm.record.Record*

Simple class to add ability to get xmlid from record itself

as_xmlid(*module=None*)

Get xmlid for record

Parameters **module** (*str*) – module to search xmlid in

Returns xmlid for this record or False

Return type str

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

X