Webservice api

Odoo is usually extended internally via modules, but many of its features and all of its data are also available from the outside for external analysis or integration with various tools. Part of the [Model Reference](https://www.odoo.com/documentation/10.0/reference/orm.html#reference-orm-model) API is easily available over [XML-RPC](http://en.wikipedia.org/wiki/XML-RPC) and accessible from a variety of languages.

**Connection**

**Configuration**

If you already have an Odoo server installed, you can just use its parameters

Warning

For Odoo Online instances (<domain>.odoo.com), users are created without a *local* password (as a person you are logged in via the Odoo Online authentication system, not by the instance itself). To use XML-RPC on Odoo Online instances, you will need to set a password on the user account you want to use:

* Log in your instance with an administrator account
* Go to Settings ‣ Users ‣ Users
* Click on the user you want to use for XML-RPC access
* Click the Change Password button
* Set a New Password value then click Change Password.

The *server url* is the instance's domain (e.g. *https://mycompany.odoo.com*), the *database name* is the name of the instance (e.g. *mycompany*). The *username* is the configured user's login as shown by the *Change Password* screen.

* Python
* Ruby
* PHP
* Java

url = <insert server URL>

db = <insert database name>

username = 'admin'

password = <insert password for your admin user (default: admin)>

**demo**

To make exploration simpler, you can also ask <https://demo.odoo.com> for a test database:

* Python
* Ruby
* PHP
* Java

import xmlrpclib

info = xmlrpclib.ServerProxy('https://demo.odoo.com/start').start()

url, db, username, password = \

info['host'], info['database'], info['user'], info['password']

**Logging in**

Odoo requires users of the API to be authenticated before they can query most data.

The xmlrpc/2/common endpoint provides meta-calls which don't require authentication, such as the authentication itself or fetching version information. To verify if the connection information is correct before trying to authenticate, the simplest call is to ask for the server's version. The authentication itself is done through the authenticate function and returns a user identifier (uid) used in authenticated calls instead of the login.

* Python
* Ruby
* PHP
* Java

common = xmlrpclib.ServerProxy('{}/xmlrpc/2/common'.format(url))

common.version()

{

"server\_version": "8.0",

"server\_version\_info": [8, 0, 0, "final", 0],

"server\_serie": "8.0",

"protocol\_version": 1,

}

* Python
* Ruby
* PHP
* Java

uid = common.authenticate(db, username, password, {})

**Calling methods**

The second endpoint is xmlrpc/2/object, is used to call methods of odoo models via the execute\_kw RPC function.

Each call to execute\_kw takes the following parameters:

* the database to use, a string
* the user id (retrieved through authenticate), an integer
* the user's password, a string
* the model name, a string
* the method name, a string
* an array/list of parameters passed by position
* a mapping/dict of parameters to pass by keyword (optional)

For instance to see if we can read the res.partner model we can call check\_access\_rights with operation passed by position and raise\_exception passed by keyword (in order to get a true/false result rather than true/error):

* Python
* Ruby
* PHP
* Java

models = xmlrpclib.ServerProxy('{}/xmlrpc/2/object'.format(url))

models.execute\_kw(db, uid, password,

'res.partner', 'check\_access\_rights',

['read'], {'raise\_exception': False})

true

**List records**

Records can be listed and filtered via [search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search).

[search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search) takes a mandatory [domain](https://www.odoo.com/documentation/10.0/reference/orm.html#reference-orm-domains) filter (possibly empty), and returns the database identifiers of all records matching the filter. To list customer companies for instance:

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password,

'res.partner', 'search',

[[['is\_company', '=', True], ['customer', '=', True]]])

[7, 18, 12, 14, 17, 19, 8, 31, 26, 16, 13, 20, 30, 22, 29, 15, 23, 28, 74]

**Pagination**

By default a search will return the ids of all records matching the condition, which may be a huge number. offset and limit parameters are available to only retrieve a subset of all matched records.

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password,

'res.partner', 'search',

[[['is\_company', '=', True], ['customer', '=', True]]],

{'offset': 10, 'limit': 5})

[13, 20, 30, 22, 29]

**Count records**

Rather than retrieve a possibly gigantic list of records and count them, [search\_count()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search_count) can be used to retrieve only the number of records matching the query. It takes the same [domain](https://www.odoo.com/documentation/10.0/reference/orm.html#reference-orm-domains) filter as [search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search) and no other parameter.

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password,

'res.partner', 'search\_count',

[[['is\_company', '=', True], ['customer', '=', True]]])

19

Warning

calling search then search\_count (or the other way around) may not yield coherent results if other users are using the server: stored data could have changed between the calls

**Read records**

Record data is accessible via the [read()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.read) method, which takes a list of ids (as returned by [search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search)) and optionally a list of fields to fetch. By default, it will fetch all the fields the current user can read, which tends to be a huge amount.

* Python
* Ruby
* PHP
* Java

ids = models.execute\_kw(db, uid, password,

'res.partner', 'search',

[[['is\_company', '=', True], ['customer', '=', True]]],

{'limit': 1})

[record] = models.execute\_kw(db, uid, password,

'res.partner', 'read', [ids])

# count the number of fields fetched by default

len(record)

121

Conversedly, picking only three fields deemed interesting.

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password,

'res.partner', 'read',

[ids], {'fields': ['name', 'country\_id', 'comment']})

[{"comment": false, "country\_id": [21, "Belgium"], "id": 7, "name": "Agrolait"}]

Note

even if the id field is not requested, it is always returned

**Listing record fields**

[fields\_get()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.fields_get) can be used to inspect a model's fields and check which ones seem to be of interest.

Because it returns a large amount of meta-information (it is also used by client programs) it should be filtered before printing, the most interesting items for a human user are string (the field's label), help (a help text if available) and type (to know which values to expect, or to send when updating a record):

* Python
* Ruby
* PHP
* Java

models.execute\_kw(

db, uid, password, 'res.partner', 'fields\_get',

[], {'attributes': ['string', 'help', 'type']})

{

"ean13": {

"type": "char",

"help": "BarCode",

"string": "EAN13"

},

"property\_account\_position\_id": {

"type": "many2one",

"help": "The fiscal position will determine taxes and accounts used for the partner.",

"string": "Fiscal Position"

},

"signup\_valid": {

"type": "boolean",

"help": "",

"string": "Signup Token is Valid"

},

"date\_localization": {

"type": "date",

"help": "",

"string": "Geo Localization Date"

},

"ref\_company\_ids": {

"type": "one2many",

"help": "",

"string": "Companies that refers to partner"

},

"sale\_order\_count": {

"type": "integer",

"help": "",

"string": "# of Sales Order"

},

"purchase\_order\_count": {

"type": "integer",

"help": "",

"string": "# of Purchase Order"

},

**Search and read**

Because it is a very common task, Odoo provides a search\_read() shortcut which as its name suggests is equivalent to a [search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search) followed by a [read()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.read), but avoids having to perform two requests and keep ids around.

Its arguments are similar to [search()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.search)'s, but it can also take a list of fields (like [read()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.read), if that list is not provided it will fetch all fields of matched records):

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password,

'res.partner', 'search\_read',

[[['is\_company', '=', True], ['customer', '=', True]]],

{'fields': ['name', 'country\_id', 'comment'], 'limit': 5})

[

{

"comment": false,

"country\_id": [ 21, "Belgium" ],

"id": 7,

"name": "Agrolait"

},

{

"comment": false,

"country\_id": [ 76, "France" ],

"id": 18,

"name": "Axelor"

},

{

"comment": false,

"country\_id": [ 233, "United Kingdom" ],

"id": 12,

"name": "Bank Wealthy and sons"

},

{

"comment": false,

"country\_id": [ 105, "India" ],

"id": 14,

"name": "Best Designers"

},

{

"comment": false,

"country\_id": [ 76, "France" ],

"id": 17,

"name": "Camptocamp"

}

]

**Create records**

Records of a model are created using [create()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.create). The method will create a single record and return its database identifier.

[create()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.create) takes a mapping of fields to values, used to initialize the record. For any field which has a default value and is not set through the mapping argument, the default value will be used.

* Python
* Ruby
* PHP
* Java

id = models.execute\_kw(db, uid, password, 'res.partner', 'create', [{

'name': "New Partner",

}])

78

Warning

while most value types are what would be expected (integer for [Integer](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Integer), string for [Char](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Char) or [Text](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Text)),

* [Date](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Date), [Datetime](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Datetime) and Binary fields use string values
* [One2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.One2many) and [Many2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Many2many) use a special command protocol detailed in [the documentation to the write method](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.write).

**Update records**

Records can be updated using [write()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.write), it takes a list of records to update and a mapping of updated fields to values similar to [create()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.create).

Multiple records can be updated simultanously, but they will all get the same values for the fields being set. It is not currently possible to perform "computed" updates (where the value being set depends on an existing value of a record).

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password, 'res.partner', 'write', [[id], {

'name': "Newer partner"

}])

# get record name after having changed it

models.execute\_kw(db, uid, password, 'res.partner', 'name\_get', [[id]])

[[78, "Newer partner"]]

**Delete records**

Records can be deleted in bulk by providing their ids to [unlink()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.unlink).

* Python
* Ruby
* PHP
* Java

models.execute\_kw(db, uid, password, 'res.partner', 'unlink', [[id]])

# check if the deleted record is still in the database

models.execute\_kw(db, uid, password,

'res.partner', 'search', [[['id', '=', id]]])

[]

**Inspection and introspection**

While we previously used [fields\_get()](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.models.Model.fields_get) to query a model and have been using an arbitrary model from the start, Odoo stores most model metadata inside a few meta-models which allow both querying the system and altering models and fields (with some limitations) on the fly over XML-RPC.

**ir.model**

Provides information about Odoo models via its various fields

name

a human-readable description of the model

model

the name of each model in the system

state

whether the model was generated in Python code (base) or by creating an ir.model record (manual)

field\_id

list of the model's fields through a [One2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.One2many) to [ir.model.fields](https://www.odoo.com/documentation/10.0/api_integration.html#reference-webservice-inspection-fields)

view\_ids

[One2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.One2many) to the [Views](https://www.odoo.com/documentation/10.0/reference/views.html#reference-views) defined for the model

access\_ids

[One2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.One2many) relation to the [Access Control](https://www.odoo.com/documentation/10.0/reference/security.html#reference-security-acl) set on the model

ir.model can be used to

* query the system for installed models (as a precondition to operations on the model or to explore the system's content)
* get information about a specific model (generally by listing the fields associated with it)
* create new models dynamically over RPC

Warning

* "custom" model names must start with x\_
* the state must be provided and manual, otherwise the model will not be loaded
* it is not possible to add new *methods* to a custom model, only fields

a custom model will initially contain only the "built-in" fields available on all models:

* Python
* PHP
* Ruby
* Java

models.execute\_kw(db, uid, password, 'ir.model', 'create', [{

'name': "Custom Model",

'model': "x\_custom\_model",

'state': 'manual',

}])

models.execute\_kw(

db, uid, password, 'x\_custom\_model', 'fields\_get',

[], {'attributes': ['string', 'help', 'type']})

{

"create\_uid": {

"type": "many2one",

"string": "Created by"

},

"create\_date": {

"type": "datetime",

"string": "Created on"

},

"\_\_last\_update": {

"type": "datetime",

"string": "Last Modified on"

},

"write\_uid": {

"type": "many2one",

"string": "Last Updated by"

},

"write\_date": {

"type": "datetime",

"string": "Last Updated on"

},

"display\_name": {

"type": "char",

"string": "Display Name"

},

"id": {

"type": "integer",

"string": "Id"

}

}

**ir.model.fields**

Provides information about the fields of Odoo models and allows adding custom fields without using Python code

model\_id

[Many2one](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Many2one) to [ir.model](https://www.odoo.com/documentation/10.0/api_integration.html#reference-webservice-inspection-models) to which the field belongs

name

the field's technical name (used in read or write)

field\_description

the field's user-readable label (e.g. string in fields\_get)

ttype

the [type](https://www.odoo.com/documentation/10.0/reference/orm.html#reference-orm-fields) of field to create

state

whether the field was created via Python code (base) or via ir.model.fields (manual)

required, readonly, translate

enables the corresponding flag on the field

groups

[field-level access control](https://www.odoo.com/documentation/10.0/reference/security.html#reference-security-fields), a [Many2many](https://www.odoo.com/documentation/10.0/reference/orm.html#odoo.fields.Many2many) to res.groups

selection, size, on\_delete, relation, relation\_field, domain

type-specific properties and customizations, see [the fields documentation](https://www.odoo.com/documentation/10.0/reference/orm.html#reference-orm-fields) for details

Like custom models, only new fields created with state="manual" are activated as actual fields on the model.

Warning

computed fields can not be added via ir.model.fields, some field meta-information (defaults, onchange) can not be set either

* Python
* PHP
* Ruby
* Java

id = models.execute\_kw(db, uid, password, 'ir.model', 'create', [{

'name': "Custom Model",

'model': "x\_custom",

'state': 'manual',

}])

models.execute\_kw(

db, uid, password,

'ir.model.fields', 'create', [{

'model\_id': id,

'name': 'x\_name',

'ttype': 'char',

'state': 'manual',

'required': True,

}])

record\_id = models.execute\_kw(

db, uid, password,

'x\_custom', 'create', [{

'x\_name': "test record",

}])

models.execute\_kw(db, uid, password, 'x\_custom', 'read', [[record\_id]])

[

{

"create\_uid": [1, "Administrator"],

"x\_name": "test record",

"\_\_last\_update": "2014-11-12 16:32:13",

"write\_uid": [1, "Administrator"],

"write\_date": "2014-11-12 16:32:13",

"create\_date": "2014-11-12 16:32:13",

"id": 1,

"display\_name": "test record"

}

]

**Workflow manipulations**

[Workflows](https://www.odoo.com/documentation/10.0/reference/workflows.html#reference-workflows) can be moved along by sending them *signals*. Instead of using the top-level execute\_kw, signals are sent using exec\_workflow.

Signals are sent to a specific record, and possibly trigger a transition on the workflow instance associated with the record.

Warning

this example needs account module installed

* Python
* PHP
* Ruby
* Java

client = models.execute\_kw(

db, uid, password,

'res.partner', 'search\_read',

[[('customer', '=', True)]],

{'limit': 1, 'fields': [

'property\_account\_receivable\_id',

'property\_payment\_term\_id',

'property\_account\_position\_id']

})[0]

invoice\_id = models.execute\_kw(

db, uid, password,

'account.invoice', 'create', [{

'partner\_id': client['id'],

'account\_id': client['property\_account\_receivable\_id'][0],

'invoice\_line\_ids': [(0, False, {'name': "AAA"})]

}])

models.exec\_workflow(

db, uid, password, 'account.invoice', 'invoice\_open', invoice\_id)

**Report printing**

Available reports can be listed by searching the ir.actions.report.xml model, fields of interest being

model

the model on which the report applies, can be used to look for available reports on a specific model

name

human-readable report name

report\_name

the technical name of the report, used to print it

Reports can be printed over RPC with the following information:

* the name of the report (report\_name)
* the ids of the records to include in the report
* Python
* PHP
* Ruby
* Java

invoice\_ids = models.execute\_kw(

db, uid, password, 'account.invoice', 'search',

[[('type', '=', 'out\_invoice'), ('state', '=', 'open')]])

report = xmlrpclib.ServerProxy('{}/xmlrpc/2/report'.format(url))

result = report.render\_report(

db, uid, password, 'account.report\_invoice', invoice\_ids)

report\_data = result['result'].decode('base64')

Note

the report is sent as PDF binary data encoded in [base64](http://en.wikipedia.org/wiki/Base64), it must be decoded and may need to be saved to disk before use