# External 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 (../reference/orm.html#reference-orm-model) API is easily available over XML-RPC (https://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 3** 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 (https://demo.odoo.com) for a test database:

**Python 3** Ruby PHP Java

```
import xmlrpc.client
info = xmlrpc.client.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 3** Ruby PHP Java

```
common = xmlrpc.client.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 3** 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 3** Ruby PHP Java

```
models = xmlrpc.client.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()` (../reference/orm.html#odoo.models.Model.search)](../reference/orm.html#odoo.models.Model.search).

[`search()` (../reference/orm.html#odoo.models.Model.search)](../reference/orm.html#odoo.models.Model.search) takes a mandatory [domain (../reference/orm.html#reference-orm-domains)](../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 3** 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 3** 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()`** (../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 (../reference/orm.html#reference-orm-domains) filter as **`search()`** (../reference/orm.html#odoo.models.Model.search) and no other parameter.

**Python 3** 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()`** (../reference/orm.html#odoo.models.Model.read) method, which takes a list of ids (as returned by **`search()`** (../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 3** 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 3** 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": "Agro
```

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

## Listing record fields

**fields_get()** (../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 3** 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 us
        "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()` (../reference/orm.html#odoo.models.Model.search) followed by a `read()` (../reference/orm.html#odoo.models.Model.read), but avoids having to perform two requests and keep ids around.

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

**Python 3** 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()`** (../reference/orm.html#odoo.models.Model.create). The method will create a single record and return its database identifier.

**`create()`** (../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 3** 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`** **(../reference/orm.html#odoo.fields.Integer)**, string for **`Char`** **(../reference/orm.html#odoo.fields.Char)** or **`Text`** **(../reference/orm.html#odoo.fields.Text)**),

- **`Date`** **(../reference/orm.html#odoo.fields.Date)**, **`Datetime`** **(../reference/orm.html#odoo.fields.Datetime)** and **`Binary`** fields use string values
- **`One2many`** **(../reference/orm.html#odoo.fields.One2many)** and **`Many2many`** **(../reference/orm.html#odoo.fields.Many2many)** use a special command protocol detailed in **`the documentation to the write method`** **(../reference/orm.html#odoo.models.Model.write)**.

## Update records

Records can be updated using **`write()`** (../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()`** (../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 3** 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()** (../reference/orm.html#odoo.models.Model.unlink).

**Python 3** 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()** (../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** (../reference/orm.html#odoo.fields.One2many) to ir.model.fields

**`view_ids`**
: **One2many** (../reference/orm.html#odoo.fields.One2many) to the Views (../reference/views.html#reference-views) defined for the model

**`access_ids`**

`One2many` (../reference/orm.html#odoo.fields.One2many) relation to the Access Control (../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 3** 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`** (../reference/orm.html#odoo.fields.Many2one) to ir.model 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 (../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 (../reference/security.html#reference-security-fields), a **`Many2many`** (../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 (../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 3** 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"
    }
]
```