ERPpeek Documentation

Release 1.6.3

Florent Xicluna

December 30, 2015

Contents

A versatile tool for browsing Odoo / OpenERP data

The ERPpeek library communicates with any Odoo / OpenERP server (>= 5.0) using the standard XML-RPC interface.

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

The Introduction describes its primary uses as a *command line tool* or within an *interactive shell*.

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

Contents:

CHAPTER 1

Introduction

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

1.1 Installation

Download and install the latest release from PyPI:

```
pip install -U erppeek
```

1.2 Command line arguments

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

```
$ erppeek --help
Usage: erppeek [options] [search_term_or_id [search_term_or_id ...]]

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

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -l, --list            list sections of the configuration
  --env=ENV             read connection settings from the given section
  -c CONFIG, --config=CONFIG
                        specify alternate config file (default: 'erppeek.ini')
  --server=SERVER       full URL to the XML-RPC server (default: http://localhost:8069)
  -d DB, --db=DB        database
  -u USER, --user=USER  username
  -p PASSWORD, --password=PASSWORD
                        password, or it will be requested on login
  -m MODEL, --model=MODEL
                        the type of object to find
  -f FIELDS, --fields=FIELDS
                        restrict the output to certain fields (multiple allowed)
  -i, --interact        use interactively; default when no model is queried
  -v, --verbose         verbose
$ #
```

Example:

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

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

1.3 Interactive use

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

```
[DEFAULT]
scheme = http
host = localhost
port = 8069
database = openerp
username = admin
options = -c /path/to/openerp-server.conf --without-demo all

[demo]
username = demo
password = demo

[local]
scheme = local
```

Connect to the Odoo server:

```
erppeek --list
erppeek --env demo
```

This is a sample session:

```
>>> model('res.users')
<Model 'res.users'>
>>> model('res.users').count()
4
>>> model('ir.cron').read(['active = False'], 'active function')
[{'active': False, 'function': 'run_mail_scheduler', 'id': 1},
 {'active': False, 'function': 'run_bdr_scheduler', 'id': 2},
 {'active': False, 'function': 'scheduled_fetch_new_scans', 'id': 9}]
>>> #
>>> client.modules('delivery')
{'uninstalled': ['delivery', 'sale_delivery_report']}
>>> client.upgrade('base')
1 module(s) selected
```

```
42 module(s) to process:
  to upgrade    account
  to upgrade    account_chart
  to upgrade    account_tax_include
  to upgrade    base
  ...
>>> #
```

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

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

More details in the Tutorial section.

CHAPTER 2

ERPpeek API

The library provides few objects to access the OpenObject model and the associated services provided by the Odoo XML-RPC API.

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

2.1 Client and Services

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

class `erppeek.`**`Client`**(*server*, *db=None*, *user=None*, *password=None*, *transport=None*, *verbose=False*)

Connection to an Odoo instance.

This is the top level object. The *server* is the URL of the instance, like `http://localhost:8069`. If *server* is an `openerp` module, it is used to connect to the local server (>= 6.1).

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

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

Create a connection to a defined environment.

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

`Client.`**`create_database`**(*passwd*, *database*, *demo=False*, *lang='en_US'*, *user_password='admin'*)

Create a new database.

The superadmin *passwd* and the *database* name are mandatory. By default, *demo* data are not loaded and *lang* is `en_US`. Wait for the thread to finish and login if successful.

Client.**login**(*user*, *password=None*, *database=None*)
Switch *user* and (optionally) *database*.

If the *password* is not available, it will be asked.

Client.**context**
Default context used for all the methods (default `None`). In *interactive mode*, this default context contains the language of the shell environment (variable `LANG`). Do not update the context, either copy it or replace it:

```
# Set language to German
client.context = {'lang': 'de_DE', 'preferred_color': 'blue'}
# ... do something

# Switch to Italian
client.context = dict(client.context, lang='it_IT')
# ... do something

# and AVOID (because it changes the context of existing records)
client.context['lang'] = 'fr_FR'
```

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

Note: In *interactive mode*, when connected to the local Odoo server, the *get_pool(db_name=None)* function helps to grab a model registry for the current database. The cursor factory is available on the registry as `get_pool().cursor()` (Odoo) or `get_pool().db.cursor()` (OpenERP <= 7).

2.1.1 Objects

Client.**search**(*obj*, *domain*, *offset=0*, *limit=None*, *order=None*, *context=None*)
Filter the records in the *domain*, return the `ids`.

Client.**count**(*obj*, *domain*, *context=None*)
Count the records in the *domain*.

Client.**read**(*obj*, *domain*, *fields=None*, *offset=0*, *limit=None*, *order=None*, *context=None*)
Wrapper for `client.execute(obj, 'read', [...], ('a', 'b'))`.

The first argument *obj* is the model name (example: `"res.partner"`)

The second argument, *domain*, accepts:

- `[('name', '=', 'mushroom'), ('state', '!=', 'draft')]`
- `['name = mushroom', 'state != draft']`
- `[]`
- a list of ids `[1, 2, 3]` or a single id `42`

The third argument, *fields*, accepts:

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

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

If *domain* is a single id, then:

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

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

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

`Client.`**`perm_read`**(*obj*, *ids*, *context=None*, *details=True*)

Lookup metadata about the records in the *ids* list. Return a list of dictionaries with the following keys:

- `id`: object id
- `create_uid`: user who created the record
- `create_date`: date when the record was created
- `write_uid`: last user who changed the record
- `write_date`: date of the last change to the record
- `xmlid`: XML ID to use to refer to this record (if there is one), in format `module.name` (not available with OpenERP 5)

If *details* is True, the `create_uid` and `write_uid` contain the name of the user.

`Client.`**`write`**(*obj*, *ids*, *values*, *context=None*)

Update the record(s) with the content of the *values* dictionary.

`Client.`**`create`**(*obj*, *values*, *context=None*)

Create a new record for the model. The argument *values* is a dictionary of values for the new record. Return the object `id`.

`Client.`**`copy`**(*obj*, *id*, *default=None*, *context=None*)

Copy a record and return the new `id`. The optional argument *default* is a mapping which overrides some values of the new record.

`Client.`**`unlink`**(*obj*, *ids*, *context=None*)

Delete records with the given *ids*

`Client.`**`models`**(*name=''*)

Return a dictionary of models.

The argument *name* is a pattern to filter the models returned. If omitted, all models are returned. Keys are camel case names of the models. Values are instances of *`Model`*.

The return value can be used to declare the models in the global namespace:

```
>>> globals().update(client.models('res.'))
```

`Client.`**`model`**(*name*, *check=True*)

Return a *`Model`* instance.

The argument *name* is the name of the model. If the optional argument *check* is `False`, no validity check is done.

`Client.`**`keys`**(*obj*)

Wrapper for *`Model.keys()`* method.

Client.**fields**(*obj*, *names=None*)
Wrapper for *Model.fields()* method.

Client.**field**(*obj*, *name*)
Wrapper for *Model.field()* method.

Client.**access**(*obj*, *mode='read'*)
Wrapper for *Model.access()* method.

2.1.2 Advanced methods

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

Client.**execute**(*obj*, *method*, **params*, ***kwargs*)
Wrapper around `object.execute` RPC method.

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

Client.**execute_kw**(*obj*, *ids*, *params*, *kwargs=None*)
Wrapper around `object.execute_kw` RPC method.

Does not exist if server is OpenERP 5.

Client.**exec_workflow**(*obj*, *signal*, *obj_id*)
Wrapper around `object.exec_workflow` RPC method.

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

Client.**report**(*obj*, *ids*, *datas=None*, *context=None*)
Wrapper around `report.report` RPC method.

Client.**render_report**(*obj*, *ids*, *datas=None*, *context=None*)
Wrapper around `report.render_report` RPC method.

Does not exist if server is OpenERP 5.

Client.**report_get**(*report_id*)
Wrapper around `report.report_get` RPC method.

Client.**wizard**(*name*, *datas=None*, *action='init'*, *context=None*)
Wrapper around `wizard.create` and `wizard.execute` RPC methods.

If only *name* is provided, a new wizard is created and its `id` is returned. If *action* is not `"init"`, then the action is executed. In this case the *name* is either an `id` or a string. If the *name* is a string, the wizard is created before the execution. The optional *datas* argument provides data for the action. The optional *context* argument is passed to the RPC method.

Removed in OpenERP 7.

2.1.3 XML-RPC Services

The nake XML-RPC services are exposed too. There are five services. The *db* and the *common* services expose few methods which might be helpful for server administration. Use the `dir()` function to introspect them. The three other services should not be used directly: they are in the private namespace, starting with _ because their methods are wrapped and exposed on the *Client* object itself. Please refer to the Odoo documentation for more details.

`Client.`**`db`**
Expose the `db` *Service*.

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

`Client.`**`common`**
Expose the `common` *Service*.

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

`Client.`**`_object`**
Expose the `object` *Service*.

`Client.`**`_report`**
Expose the `report` *Service*.

`Client.`**`_wizard`**
Expose the `wizard` *Service*.

Removed in OpenERP 7.

class `erppeek.`**`Service`**(*server*, *endpoint*, *methods*, *transport=None*, *verbose=False*)
A wrapper around XML-RPC endpoints.

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

2.1.4 Manage addons

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

`Client.`**`modules`**(*name=''*, *installed=None*)
Return a dictionary of modules.

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

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

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

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

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

2.2 Model and Records

In addition to the thin wrapper methods, the *Client* provides a high level API which encapsulates objects into Active Records.

The *Model* is instantiated using the *Client.model()* method or directly through camel case attributes.

Example: both `client.model('res.company')` and `client.ResCompany` return the same *Model*.

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

keys()
Return the keys of the model.

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

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

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

access(*mode='read'*)
Check if the user has access to this model.

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

browse(*domain*, *offset=0*, *limit=None*, *order=None*, *context=None*)
Return a *Record* or a *RecordList*.

The argument *domain* accepts a single integer `id`, a list of ids or a search domain. If it is a single integer, the return value is a *Record*. Otherwise, the return value is a *RecordList*. Be careful when passing a list of ids, because an empty list will be considered an empty domain and will find all records in the database.

get(*domain*, *context=None*)
Return a single *Record*.

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

create(*values*, *context=None*)
Create a *Record*.

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

The newly created *Record* is returned.

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

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

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

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

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

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

perm_read(*context=None*)
Wrapper for the *Record.perm_read()* method.

write(*values*, *context=None*)
Wrapper for the *Record.write()* method.

unlink(*context=None*)
Wrapper for the *Record.unlink()* method.

_external_id
Retrieve the External IDs of the *RecordList*.

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

class erppeek.**Record**(*model*, *id*)
A class for all Odoo records.

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

_external_id
Retrieve the External ID of the *Record*.

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

_send(*signal*)
Trigger workflow *signal* for this *Record*.

copy(*default=None*, *context=None*)
Copy a record and return the new *Record*.

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

perm_read(*context=None*)
Read the metadata of the *Record*.

Return a dictionary of values. See *Client.perm_read()* for details.

read(*fields=None*, *context=None*)
Read the *fields* of the *Record*.

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

refresh()
Force refreshing the record's data.

unlink(*context=None*)
Delete the current *Record* from the database.

write(*values*, *context=None*)
Write the *values* in the *Record*.

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

2.3 Utilities

erppeek.**lowercase**(*s*)
Convert to lowercase with dots.

```
>>> lowercase('ResCompany')
'res.company'
```

erppeek.**mixedcase**(*s*)
Convert to MixedCase.

```
>>> mixedcase('res.company')
'ResCompany'
```

erppeek.**issearchdomain**(*arg*)
Check if the argument is a search domain.

Examples:

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

erppeek.**searchargs**(*params*, *kwargs=None*, *context=None*)
Compute the 'search' parameters.

erppeek.**format_exception**(*type*, *value*, *tb*, *limit=None*, *chain=True*)
Format a stack trace and the exception information.

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

erppeek.**read_config**(*section=None*)
Read the environment settings from the configuration file.

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

erppeek.**start_odoo_services**(*options=None*, *appname=None*)
Initialize the Odoo services.

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

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

Return the `openerp` package.

CHAPTER 3

Tutorial

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

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

Steps:

3.1 First connection

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

Open the ERPpeek shell:

```
$ erppeek
```

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

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

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

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

```
$ erppeek
Usage (some commands):
    models(name)                    # List models matching pattern
    model(name)                     # Return a Model instance
    model(name).keys()              # List field names of the model
    model(name).fields(names=None)  # Return details for the fields
    model(name).field(name)         # Return details for the field
    model(name).browse(domain)
    model(name).browse(domain, offset=0, limit=None, order=None)
```

```
                                    # Return a RecordList

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

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

As we'll see later, the most interesting method here is probably *model()* which returns a *Model* object with nice wrappers.

And it confirms that the default database is not available:

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

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

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

3.2 Create a database

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

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

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

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

```
[DEFAULT]
host = localhost
port = 8069
```

```
[demo]
database = demo
username = joe
```

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

3.3 Find the users

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

Where is the table for the users?

```
>>> client
<Client 'http://localhost:8069#demo'>
>>> models('user')
{'ResUsers': <Model 'res.users'>, 'ResWidgetUser': <Model 'res.widget.user'>}
```

We've listed two models which matches the name, `res.users` and `res.widget.user`. We reach the users' model using the *`model()`* method and we want to introspect its fields. Fortunately, the *`Model`* class provides methods to retrieve all the details.

```
>>> model('res.users')
<Model 'res.users'>
>>> print(model('res.users').keys())
['action_id', 'active', 'company_id', 'company_ids', 'context_lang',
 'context_tz', 'date', 'groups_id', 'id', 'login', 'menu_id', 'menu_tips',
 'name', 'new_password', 'password', 'signature', 'user_email', 'view']
>>> model('res.users').field('view')
{'digits': [16, 2],
 'fnct_inv': '_set_interface_type',
 'fnct_inv_arg': False,
 'fnct_search': False,
 'func_obj': False,
 'function': '_get_interface_type',
 'help': 'OpenERP offers a simplified and an extended user interface. If\
 you use OpenERP for the first time we strongly advise you to select the\
 simplified interface, which has less features but is easier to use. You\
 can switch to the other interface from the User/Preferences menu at any\
 time.',
 'selection': [['simple', 'Simplified'], ['extended', 'Extended']],
 'store': False,
 'string': 'Interface',
 'type': 'selection'}
>>> #
```

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

```
>>> model('res.users').count()
1
>>> admin_user = model('res.users').browse(1)
>>> admin_user.groups_id
<RecordList 'res.groups,[1, 2, 3]'>
>>> admin_user.groups_id.name
['Access Rights', 'Configuration', 'Employee']
>>> admin_user.groups_id.full_name
```

```
['Administration / Access Rights',
 'Administration / Configuration',
 'Human Resources / Employee']
>>> admin_user.perm_read()
{'create_date': False,
 'create_uid': False,
 'id': 1,
 'write_date': '2012-09-01 09:01:36.631090',
 'write_uid': [1, 'Administrator'],
 'xmlid': 'base.user_admin'}
```

3.4 Create a new user

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

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

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

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

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

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

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

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

Success!

3.5 Explore the model

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

```
>>> client.user
'joe'
```

```
>>> all_models = sorted(models().values(), key=str)
>>> len(all_models)
92
```

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

```
>>> # Read-only models
>>> len([m for m in all_models if not m.access('write')])
44
>>> #
>>> # Writable but cannot delete
>>> [m for m in all_models if m.access('write') and not m.access('unlink')]
[<Model 'ir.property'>]
>>> #
>>> # Unreadable models
>>> [m for m in all_models if not m.access('read')]
[<Model 'ir.actions.todo'>,
 <Model 'ir.actions.todo.category'>,
 <Model 'res.payterm'>]
>>> #
>>> # Now print the number of entries in all (readable) models
>>> for m in all_models:
...     mcount = m.access() and m.count()
...     if not mcount:
...         continue
...     print('%4d  %s' % (mcount, m))
...
  81  <Model 'ir.actions.act_window'>
  14  <Model 'ir.actions.act_window.view'>
  85  <Model 'ir.actions.act_window_close'>
  85  <Model 'ir.actions.actions'>
   4  <Model 'ir.actions.report.xml'>
   3  <Model 'ir.config_parameter'>
   2  <Model 'ir.cron'>
   1  <Model 'ir.mail_server'>
  92  <Model 'ir.model'>
 126  <Model 'ir.model.access'>
1941  <Model 'ir.model.data'>
 658  <Model 'ir.model.fields'>
  32  <Model 'ir.module.category'>
 207  <Model 'ir.module.module'>
 432  <Model 'ir.module.module.dependency'>
   8  <Model 'ir.rule'>
  63  <Model 'ir.ui.menu'>
 185  <Model 'ir.ui.view'>
   1  <Model 'ir.ui.view_sc'>
  72  <Model 'ir.values'>
   1  <Model 'res.bank'>
   1  <Model 'res.company'>
 253  <Model 'res.country'>
  51  <Model 'res.country.state'>
  48  <Model 'res.currency'>
  49  <Model 'res.currency.rate'>
   9  <Model 'res.groups'>
   1  <Model 'res.lang'>
   1  <Model 'res.partner'>
   1  <Model 'res.partner.address'>
```

```
   1  <Model 'res.partner.bank.type'>
   1  <Model 'res.partner.bank.type.field'>
   5  <Model 'res.partner.title'>
   1  <Model 'res.request.link'>
   2  <Model 'res.users'>
   6  <Model 'res.widget'>
   1  <Model 'res.widget.user'>
>>> #
>>> # Show the content of a model
>>> config_params = model('ir.config_parameter').browse([])
>>> config_params.read()
[{'id': 1, 'key': 'web.base.url', 'value': 'http://localhost:8069'},
 {'id': 2, 'key': 'database.create_date', 'value': '2012-09-01 09:01:12'},
 {'id': 3,
  'key': 'database.uuid',
  'value': '52fc9630-f49e-2222-e622-08002763afeb'}]
```

3.6 Browse the records

Query the "`res.country`" model:

```
>>> model('res.country').keys()
['address_format', 'code', 'name']
>>> model('res.country').browse(['name like public'])
<RecordList 'res.country,[41, 42, 57, 62, 116, 144]'>
>>> model('res.country').browse(['name like public']).name
['Central African Republic',
 'Congo, Democratic Republic of the',
 'Czech Republic',
 'Dominican Republic',
 'Kyrgyz Republic (Kyrgyzstan)',
 'Macedonia, the former Yugoslav Republic of']
>>> model('res.country').browse(['code > Y'], order='code ASC').read('code name')
[{'code': 'YE', 'id': 247, 'name': 'Yemen'},
 {'code': 'YT', 'id': 248, 'name': 'Mayotte'},
 {'code': 'YU', 'id': 249, 'name': 'Yugoslavia'},
 {'code': 'ZA', 'id': 250, 'name': 'South Africa'},
 {'code': 'ZM', 'id': 251, 'name': 'Zambia'},
 {'code': 'ZR', 'id': 252, 'name': 'Zaire'},
 {'code': 'ZW', 'id': 253, 'name': 'Zimbabwe'}]
>>> #
```

... the tutorial is done.

Jump to the ERPpeek API for further details.

CHAPTER 4

Developer's notes

4.1 Source code

- Source code and issue tracker on GitHub.
- Continuous tests against Python 2.6 through 3.5 and PyPy, on Travis-CI platform.

4.2 Third-party integration

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

For example:

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

4.3 Changes

4.3.1 1.6.3 (2015-12-30)

- Do not parse long integers which overflow in XML-RPC.

4.3.2 1.6.2 (2015-09-17)

- Add an optional `transport` argument to the `Client` constructor. This is useful for tweaking the SSL context or adding an optional timeout parameter.
- Implement == comparison for `RecordList` instances.
- Uninstall dependent add-ons in a single call.
- Do not install/uninstall add-ons if other actions are pending.
- Do not hang when the `Client` constructor receives invalid arguments.
- Fix `str(record)` and `print(record)` with non-ASCII names.

4.3.3 1.6.1 (2014-11-12)

- Support using `--env` and `--user` together to connect with a different user.
- Adapt for Odoo 8.0 after change `cc4fba6` on October 2014.
- Do not interpret digits with leading `0` as octal in search domain.

4.3.4 1.6 (2014-09-23)

- Compatible with Odoo 8.0.
- New attribute `Client.context` to set the default context for high-level `Model` and `Record` methods.
- Use blocking RPC call in `Client.create_database`. Asynchronous method is removed in Odoo.
- Return the interactive namespace with `main(interact=False)`. It helps to integrate with third-party libraries, such as IPython.
- Remove a duplicate `Logged in as ...` line in interactive mode.
- Remove the `search+name_get` undocumented feature which has wrong behavior when applied to an empty `RecordList`.
- Do not prevent login if access to `Client.db.list()` is denied.

4.3.5 1.6b1 (2014-06-09)

- When a function or a method fails, raise an `erppeek.Error` instead of printing a message and returning `None`.
- Switch to local mode when the command line argument points at the server configuration, like `-c path/to/openerp-server.conf`.
- Local mode compatible with Odoo trunk: support both the old and the new API.
- Use shell-like parsing for `options =` setting in local mode.
- Function `start_openerp_services` is replaced with `start_odoo_services`: it is still compatible with OpenERP 6.1 and 7 and it accepts a list of options in the first argument, similar to `sys.argv[1:]`.
- Search domains require square brackets. Usage without square brackets was deprecated since 0.5, with `UserWarning` alerts.
- Implement addition of `RecordList` of the same model.
- Drop compatibility with Python 2.5.

4.3.6 1.5.3 (2014-05-26)

- Change command line output to CSV format.
- Translate command line output according to LANG environment variable.
- Pretty print the list of modules.
- Do not report `Module(s) not found` when trying to install a module already installed.

4.3.7 1.5.2 (2014-04-12)

- Return an appropriate error message when the client is not connected.
- Two similar `Record` from different connections do not compare equal.
- Set the `PGAPPNAME` used for the PostgreSQL connection, in local mode.
- Close PostgreSQL connections on exit, in local mode.
- Implement the context manager protocol.

4.3.8 1.5.1 (2014-03-11)

- When switching to a different environment, with `Client.connect`, invalidate the previous connection to avoid mistakes (interactive mode).
- Avoid cluttering the globals in interactive mode.
- Close socket to avoid `ResourceWarning` on Python 3.
- The `get_pool` helper is only available in interactive mode and if the client is connected locally using the `openerp` package.
- Clear the last exception before entering interactive mode, only needed on Python 2.
- Fix the `searchargs` domain parser for compatibility with Python 3.4.

4.3.9 1.5 (2014-03-10)

- Advertize the `Model` and `Record` paradigm in the `usage` printed in interactive mode: it's far more easier to use, and available since 1.0.
- In interactive mode, only inject four global names: `client`, `models`, `model` and `do`. Other methods are available on `Model` and `Client` instances (`read search count keys fields access` ...).
- Always clear the `Record` cache when an arbitrary method is called on this `Record`.
- Implement == comparison for `Record` instances.
- New computed attributes `Record._external_id` and `RecordList._external_id`, and new method `Model._get_external_ids(ids=None)`.
- Better parsing of dates in search terms.
- Reject invalid == operator in search terms.
- Now the `str(...)` of a `Record` is always retrieved with `name_get`. Previously, the output was sometimes inconsistent.
- Fix `TypeError` when browsing duplicate ids.
- Fix error with `Model.get(['field = value'], context={...})`.
- Workaround an issue with some models: always pass a list of ids to `read`.
- Test the behaviour when `read` is called with a `False` id: it happens when browsing a `RecordList` for example.

4.3.10 1.4.5 (2013-03-20)

- Extend `Model.get` to retrieve a record by `xml_id`.
- Fix AttributeError when reading a mix of valid and invalid records.
- Fix `dir()` on `Record` and `RecordList` to return all declared fields, and do not report `id` field twice.
- Fix a crash with built-in OS X readline on Python 2.5 or 2.6.

4.3.11 1.4.4 (2013-03-05)

- Remove deprecated `Record.client`.
- Fix compatibility with Python 3.
- Add optional argument `check` to the `Client.model` method to bypass the verification in some cases, used to speed up the read methods.
- Do not crash when mixing non-existing and existing records: return always `False` for non-existing records.

4.3.12 1.4.3 (2013-01-10)

- Compatible with OpenERP 7.
- Set the database name as thread attribute to print it in the log file (local mode only).
- Do not try to access private methods through RPC when resolving attributes of the `Client` or any `Record` or `RecordList`.

4.3.13 1.4.2 (2012-12-19)

- Add the `get_pool` helper when connected using the `openerp` library.
- Remove the leading slash on the `server` option, if present.
- Do not try to access private methods through RPC when reading attributes of the `model(...)`.

4.3.14 1.4.1 (2012-10-05)

- Fix reading `many2one` attribute on `RecordList` object in local mode.
- Fix occasional issue on login when switching database on the same server.
- Optimization: do not propagate the call to `RecordList.write` or `RecordList.unlink` if the list is empty.
- Clear the `Record` cache on `Record._send`.
- Expose the method `Record.refresh` to clear the local cache.

4.3.15 1.4 (2012-10-01)

- New: direct connection to a local server using the `openerp` library. Use `scheme = local` and `options = -c /path/to/openerp-server.conf` in the configuration.

4.3.16 1.3.1 (2012-09-28)

- Fix method `Record._send`.

4.3.17 1.3 (2012-09-27)

- Implement exception chaining in `format_exception` to print the original traceback.
- Return a list of `Record` objects when reading the `reference` field of a `RecordList` object.
- Fix reading attributes on `RecordList` with holes or gaps.
- Accessing an empty `one2many` or `many2many` attribute on a `Record` returns a `RecordList`.
- New method `Model.get` to retrieve a single `Record`. It raises a `ValueError` if multiple records are found.
- New method `Record._send` to send a workflow signal.

4.3.18 1.2.2 (2012-09-24)

- Accept `Record` and `RecordList` attribute values when writing or creating records.
- Improve the methods `write` and `create` of `Record` and `RecordList` objects to manage `one2many` and `many2many` fields.
- Return a `Record` when reading a `reference` field. Implement the `create` and `write` methods for these fields.
- Remove undocumented alias `Record.update`.

4.3.19 1.2.1 (2012-09-21)

- Add the special operators `=ilike`, `=ilike`, `=?` and fix parsing of inequality operators >= and <=.
- Fix the `RecordList.id` attribute, and deprecate `RecordList._ids`.
- Deprecate the `Record.client` attribute: use `Record._model.client`.
- Accessing an empty `many2one` attribute on a `RecordList` now returns a `RecordList`.
- Fix `TypeError` when browsing non-existent records.

4.3.20 1.2 (2012-09-19)

- Catch some malformed search domains before sending the RPC request.
- Preserve dictionary response when calling non standard `Record` methods.
- Expose the helper `format_exception` which formats the errors received through XML-RPC.
- Support XML-RPC through HTTPS with `scheme = https` in the `erppeek.ini` configuration file.
- Print an error message when `client.upgrade(...)` does not find any module to upgrade.

4.3.21 1.1 (2012-09-04)

- When using arbitrary methods on `Record`, wrap the `id` in a list `[id]`. It fixes a recurring issue with poorly tested methods.
- Do not read all records if the `RecordList` is empty.
- Fix the bad behaviour when switching to a different database.
- Order the results when using `read` method with `order=` argument.
- Reading attributes of the sequence `<RecordList 'sea.fish,[2, 1, 2]'>` will return an ordered sequence of three items. Previously it used to return an unordered sequence of two items.
- Accept the `%(...)s` formatting for the fields parameter of the `Record.read` and the `RecordList.read` methods too.
- Add a tutorial to the documentation.

4.3.22 1.0 (2012-08-29)

- Add the test suite for Python 2 and Python 3.
- Implement `len()` for `RecordList` objects.
- Connect to the server even if the database is missing.
- Expose the method `Client.db.get_progress`.
- New method `Client.create_database` which wraps together `Client.db.create` and `Client.db.get_progress`.
- Save the readline history in `~/.erppeek_history`, only if the file already exists.
- Enable auto-completion using `rlcompleter` standard module.
- Raise an `AttributeError` when assigning to a missing or read-only attribute.

4.3.23 0.11 (2012-08-24)

- Enhance the `Model.browse()` method to accept the same keyword arguments as the `Client.search()` method.
- Fix the verbose level on `Client.connect()`.
- Fix the `Record.copy()` method.
- Fix the `Record.perm_read()` method (workaround an OpenERP bug when dealing with single ids).
- Drop the `--search` argument, because the search terms can be passed as positional arguments after the options. Explain it in the description.
- Fix the shell command. Request the password interactively if it's not in the options and not in the configuration file.

4.3.24 0.10 (2012-08-23)

- Add the `--verbose` switch to log the XML-RPC messages. Lines are truncated at 79 chars. Use `-vv` or `-vvv` to truncate at 179 or 9999 chars respectively.

- Removed the `--write` switch because it's not really useful. Use *Record.write()* or `client.write()` for example.
- Stop raising RuntimeError when calling `Client.model(name)`. Simply print the message if the name does not match.
- Fix `RecordList.read()` and `Record.read()` methods to accept the same diversity of `fields` arguments as the `Client.read()` method.
- `RecordList.read()` and `Record.read()` return instances of `RecordList` and `Record` for relational fields.
- Optimize: store the name of the `Record` when a relational field is accessed.
- Fix message wording on module install or upgrade.

4.3.25 0.9.2 (2012-08-22)

- Fix `Record.write()` and `Record.unlink()` methods.
- Fix the caching of the `Model` keys and fields and the `Record` name.

4.3.26 0.9.1 (2012-08-22)

- Fix `client.model()` method. Add `models()` to the `globals()` in interactive mode.

4.3.27 0.9 (2012-08-22)

- Add the Active Record pattern for convenience. New classes *Model*, *RecordList* and *Record*. The *Client.model()* method now returns a single *Model* instance. These models can be reached using camel case attribute too. Example: `client.model('res.company')` and `client.ResCompany` return the same *Model*.
- Refresh the list of modules before install or upgrade.
- List all modules which have `state not in ('uninstalled', 'uninstallable')` when calling `client.modules(installed=True)`.
- Add documentation.

4.3.28 0.8 (2012-04-24)

- Fix `help(client)` and `repr(...)`.
- Add basic safeguards for argument types.

4.3.29 0.7 (2012-04-04)

- Fix RuntimeError on connection.

4.3.30 0.6 (2012-04-03)

- Support Python 3.
- Return Client method instead of function when calling `client.write` or similar.
- Fix the case where *`read()`* is called with a single id.

4.3.31 0.5 (2012-03-29)

- Implement `Client.__getattr__` special attribute to call any object method, like `client.write(obj, values)`. This is somewhat redundant with `client.execute(obj, 'write', values)` and its interactive alias `do(obj, 'write', values)`.
- Add `--write` switch to enable unsafe helpers: `write`, `create`, `copy` and `unlink`.
- Tolerate domain without square brackets, but show a warning.
- Add long options `--search` for `-s`, `--interact` for `-i`.

4.3.32 0.4 (2012-03-28)

- Workaround for `sys.excepthook` ignored, related to a Python issue.

4.3.33 0.3 (2012-03-26)

- Add `--config` and `--version` switches.
- Improve documentation with session examples.
- Move the project from Launchpad to GitHub.

4.3.34 0.2 (2012-03-24)

- Allow to switch user or database: methods `client.login` and `client.connect`.
- Allow `context=` keyword argument.
- Add `access(...)` method.
- Add `%(...)s` formatting for the fields parameter of the `read(...)` method.
- Refactor the interactive mode.
- Many improvements.
- Publish on PyPI.

4.3.35 0.1 (2012-03-14)

- Initial release.
- Online documentation: http://erppeek.readthedocs.org/
- Source code and issue tracker: https://github.com/tinyerp/erppeek

CHAPTER 5

Indices and tables

- genindex
- search

CHAPTER 6

Credits

Authored and maintained by Florent Xicluna.

Derived from a script by Alan Bell.

e

Symbols

A

B

C

D

E

F

G

I

K

L

M

P

R

S

U

W