tryton-proteus Documentation

Release 6.2

Apr 18, 2022

CONTENTS

A library to access Tryton's models like a client.

CHAPTER

ONE

EXAMPLE OF USAGE

```
>>> from proteus import config, Model, Wizard, Report
```

1.1 Configuration

Configuration to connect to a sqlite memory database using trytond as module.

```
>>> config = config.set_trytond('sqlite:///:memory:')
```

There is also the *config.set_xmlrpc* method which can be used to connect using a URL, and the *config.set_xmlrpc_session* method (when used as a context manager) which connects for a session.

1.2 Activating a module

Find the module, call the activate button and run the upgrade wizard.

```
>>> Module = Model.get('ir.module')
>>> party_module, = Module.find([('name', '=', 'party')])
>>> party_module.click('activate')
>>> Wizard('ir.module.activate_upgrade').execute('upgrade')
```

1.3 Creating a party

First instanciate a new Party:

```
>>> Party = Model.get('party.party')
>>> party = Party()
>>> party.id < 0
True
```

Fill the fields:

```
>>> party.name = 'ham'
```

Save the instance into the server:

```
>>> party.save()
>>> party.name
'ham'
>>> party.id > 0
True
```

1.4 Setting the language of the party

The language on party is a *Many2One* relation field. So it requires to get a *Model* instance as value.

```
>>> Lang = Model.get('ir.lang')
>>> en, = Lang.find([('code', '=', 'en')])
>>> party.lang = en
>>> party.save()
>>> party.lang.code
'en'
```

1.5 Creating an address for the party

Addresses are store on party with a *One2Many* field. So the new address just needs to be appended to the list *addresses*.

```
>>> address = party.addresses.new(postal_code='42')
>>> party.save()
>>> party.addresses
[proteus.Model.get('party.address')(...)]
```

1.6 Adding category to the party

Categories are linked to party with a *Many2Many* field.

So first create a category

```
>>> Category = Model.get('party.category')
>>> category = Category()
>>> category.name = 'spam'
>>> category.save()
```

Append it to categories of the party

```
>>> party.categories.append(category)
>>> party.save()
>>> party.categories
[proteus.Model.get('party.category')(...)]
```

1.7 Print party label

There is a label report on *Party*.

```
>>> label = Report('party.label')
```

The report is executed with a list of records and some extra data.

```
>>> type_, data, print_, name = label.execute([party], {})
```

1.8 Sorting addresses and register order

Addresses are ordered by sequence which means they can be stored following a specific order. The *set_sequence* method stores the current order.

```
>>> address = party.addresses.new(postal_code='69')
>>> party.save()
>>> address = party.addresses.new(postal_code='23')
>>> party.save()
```

Now changing the order.

```
>>> reversed_addresses = list(reversed(party.addresses))
>>> while party.addresses:
...     _ = party.addresses.pop()
>>> party.addresses.extend(reversed_addresses)
>>> party.addresses.set_sequence()
>>> party.save()
>>> party.addresses == reversed_addresses
True
```

1.9 Setting context

Make French translatable:

```
>>> Language = Model.get('ir.lang')
>>> french, = Language.find([('code', '=', 'fr')])
>>> french.translatable = True
>>> french.save()
```

Create a category in English:

```
>>> Category = Model.get('party.category')
>>> with config.set_context(language='en'):
...     category = Category(name="Category")
...     category.save()
```

Translate in French:

```
>>> with config.set_context(language='fr'):
...     category_fr = Category(category.id)
...     category_fr.name = "Categorie"
...     category_fr.save()
```

Read in English:

```
>>> category.reload()
>>> category.name
'Category'
```

Read in French:

```
>>> category_fr.reload()
>>> category_fr.name
'Categorie'
```