**Fields**(class property)

Field types

1.Boolean :Checkbox

2.Char :TextBox

3.Text :TextArea

4.HTML

5.Integer

6.Float

7.Date :Calendar

8.DateTime

9.Binary :Image

10.Selection :DropDown

11.Reference

12.Many2one

13.One2many

14.Many2many

odoo.py scaffold New ---for creating a new module "New" from the cmd in the "addons" folder

\_rec\_name="field name"

\_order = 'date desc'

from openerp import fields

class AModel(models.Model):

\_name = 'a\_name'

name = fields.Char(

**string="Name"**,# Optional label of the field **compute="\_compute\_name\_custom"**,#Transform the fields in computed fields

**store=True,** # If computed it will store the result

**select=True,**  # Force index on field

**readonly=True,** # Field will be readonly in views

**inverse="\_write\_name"** # On update trigger

**required=True,**  # Mandatory field

**translate=True,**  # Translation enable

**help='blabla',** # Help tooltip text

**company\_dependent=True,**  # Transform columns to ir.property

**search='\_search\_function'**  # Custom search function mainly used with compute

)

# The string key is not mandatory

# by default it wil use the property name Capitalized

name = fields.Char() # Valid definition

Field inheritance

One of the new features of the API is to be able to change only one attribute of the field:

name = fields.Char(string='New Value')

Field types

1.Boolean

abool = fields.Boolean()

2.Char

Store string with variable len.:

achar = fields.Char()

Specific options:

size: data will be trimmed to specified size

translate: field can be translated

3.Text

Used to store long text.:

atext = fields.Text()

Specific options:translate

4.HTML

Used to store HTML, provides an HTML widget.:

anhtml = fields.Html()

Specific options:translate

5.Integer

Store integer value. No NULL value support. If value is not set it returns 0:

anint = fields.Integer()

6.Float

Store float value. No NULL value support. If value is not set it returns 0.0 If digits option is set it will use numeric type:

afloat = fields.Float()

afloat = fields.Float(digits=(32, 32))

afloat = fields.Float(digits=lambda cr: (32, 32))

Specific options:

digits: force use of numeric type on database. Parameter can be a tuple (int len, float len) or a callable that return a tuple and take a cursor as parameter

7.Date

Store date. The field provides some helpers:

context\_today returns current day date string based on tz

today returns current system date string

from\_string returns datetime.date() from string

to\_string returns date string from datetime.date

:

>>> from openerp import fields

>>> adate = fields.Date()

>>> fields.Date.today()

'2014-06-15'

>>> fields.Date.context\_today(self)

'2014-06-15'

>>> fields.Date.context\_today(self, timestamp=datetime.datetime.now())

'2014-06-15'

>>> fields.Date.from\_string(fields.Date.today())

datetime.datetime(2014, 6, 15, 19, 32, 17)

>>> fields.Date.to\_string(datetime.datetime.today())

'2014-06-15'

8.DateTime

Store datetime. The field provide some helper:

context\_timestamp returns current day date string based on tz

now returns current system date string

from\_string returns datetime.date() from string

to\_string returns date string from datetime.date

:

>>> fields.Datetime.context\_timestamp(self, timestamp=datetime.datetime.now())

datetime.datetime(2014, 6, 15, 21, 26, 1, 248354, tzinfo=<DstTzInfo 'Europe/Brussels' CEST+2:00:00 DST>)

>>> fields.Datetime.now()

'2014-06-15 19:26:13'

>>> fields.Datetime.from\_string(fields.Datetime.now())

datetime.datetime(2014, 6, 15, 19, 32, 17)

>>> fields.Datetime.to\_string(datetime.datetime.now())

'2014-06-15 19:26:13'

Binary

Store file encoded in base64 in bytea column:

abin = fields.Binary()

Selection

Store text in database but propose a selection widget. It induces no selection constraint in database. Selection must be set as a list of tuples or a callable that returns a list of tuples:

aselection = fields.Selection([('a', 'A')])

aselection = fields.Selection(selection=[('a', 'A')])

aselection = fields.Selection(selection='a\_function\_name')

Specific options:

selection: a list of tuple or a callable name that take recordset as input

When extending a model, if you want to add possible values to a selection field, you may use the selection\_add keyword argument:

class SomeModel(models.Model):

\_inherits = 'some.model'

type = fields.Selection(selection\_add=[('b', 'B'), ('c', 'C')])

Reference

Store an arbitrary reference to a model and a row:

aref = fields.Reference([('model\_name', 'String')])

aref = fields.Reference(selection=[('model\_name', 'String')])

aref = fields.Reference(selection='a\_function\_name')

Specific options:

selection: a list of tuple or a callable name that take recordset as input

Many2one

Store a relation against a co-model:

arel\_id = fields.Many2one('res.users')

arel\_id = fields.Many2one(comodel\_name='res.users')

an\_other\_rel\_id = fields.Many2one(comodel\_name='res.partner', delegate=True)

Specific options:

comodel\_name: name of the opposite model

delegate: set it to True to make fields of the target model accessible from the current model (corresponds to \_inherits)

One2many

Store a relation against many rows of co-model:

arel\_ids = fields.One2many('res.users', 'rel\_id')

arel\_ids = fields.One2many(comodel\_name='res.users', inverse\_name='rel\_id')

Specific options:

comodel\_name: name of the opposite model

inverse\_name: relational column of the opposite model

Many2many

Store a relation against many2many rows of co-model:

arel\_ids = fields.Many2many('res.users')

arel\_ids = fields.Many2many(comodel\_name='res.users',

relation='table\_name',

column1='col\_name',

column2='other\_col\_name')

Specific options:

comodel\_name: name of the opposite model

relation: relational table name

columns1: relational table left column name

columns2: relational table right column name

Name Conflicts

Note

fields and method name can conflict.

When you call a record as a dict it will force to look on the columns.

Fields Defaults

Default is now a keyword of a field:

You can attribute it a value or a function

name = fields.Char(default='A name')

# or

name = fields.Char(default=a\_fun)

#...

def a\_fun(self):

return self.do\_something()

Using a fun will force you to define function before fields definition.

Computed Fields

There is no more direct creation of fields.function.

Instead you add a compute kwarg. The value is the name of the function as a string or a function. This allows to have fields definition atop of class:

class AModel(models.Model):

\_name = 'a\_name'

computed\_total = fields.Float(compute='compute\_total')

def compute\_total(self):

...

self.computed\_total = x

The function can be void. It should modify record property in order to be written to the cache:

self.name = new\_value

Be aware that this assignation will trigger a write into the database. If you need to do bulk change or must be careful about performance, you should do classic call to write

To provide a search function on a non stored computed field you have to add a search kwarg on the field. The value is the name of the function as a string or a reference to a previously defined method. The function takes the second and third member of a domain tuple and returns a domain itself

def search\_total(self, operator, operand):

...

return domain # e.g. [('id', 'in', ids)]

Inverse

The inverse key allows to trigger call of the decorated function when the field is written/

Multi Fields

To have one function that compute multiple values:

@api.multi

@api.depends('field.relation', 'an\_otherfield.relation')

def \_amount(self):

for x in self:

x.total = an\_algo

x.untaxed = an\_algo

Related Field

There is not anymore fields.related fields.

Instead you just set the name argument related to your model:

participant\_nick = fields.Char(string='Nick name',

related='partner\_id.name')

The type kwarg is not needed anymore.

Setting the store kwarg will automatically store the value in database. With new API the value of the related field will be automatically updated, sweet.

participant\_nick = fields.Char(string='Nick name',

store=True,

related='partner\_id.name')

Note

When updating any related field not all translations of related field are translated if field is stored!!

Chained related fields modification will trigger invalidation of the cache for all elements of the chain.

Property Field

There is some use cases where value of the field must change depending of the current company.

To activate such behavior you can now use the company\_dependent option.

A notable evolution in new API is that are now searchable.

WIP copyable option

There is a dev running that will prevent to redefine copy by simply setting a copy option on fields:

copy=False # !! WIP to prevent redefine copy