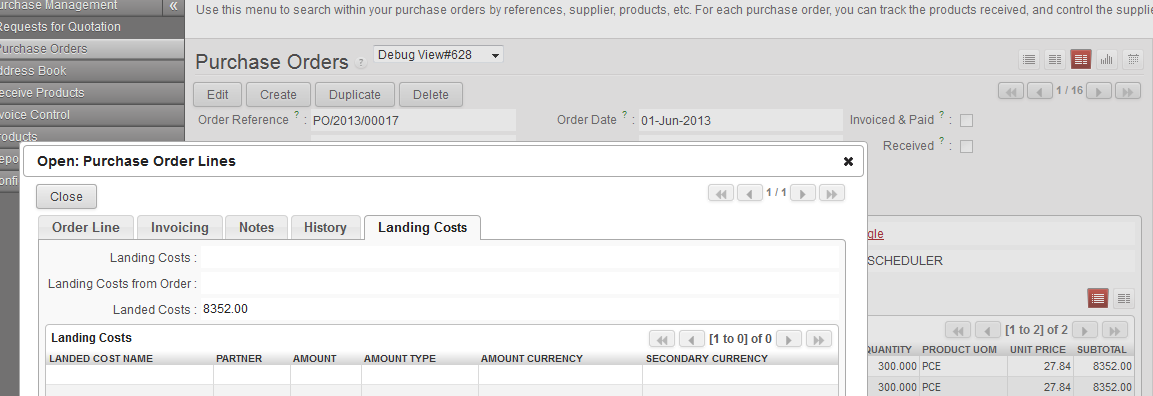
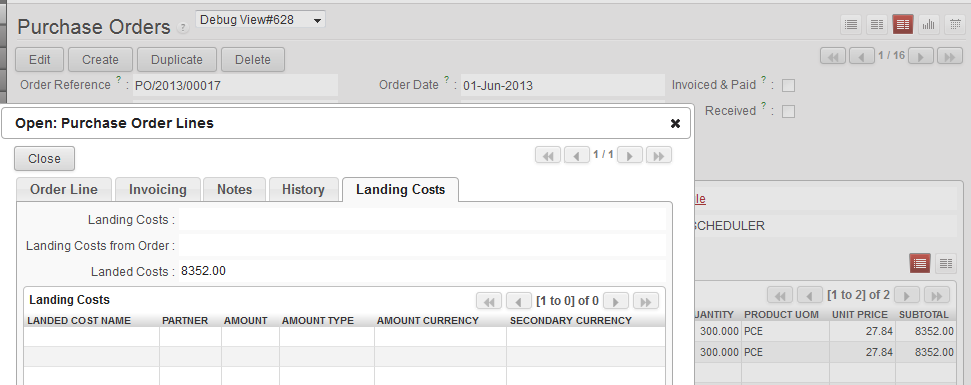
Landed cost module: purchase\_landed\_cost

Reporting of what seems to be an error when purchase order has more than 1 line. Landed costs data are shipping are wrong and **lead to a wrong Moving Average Price in product**

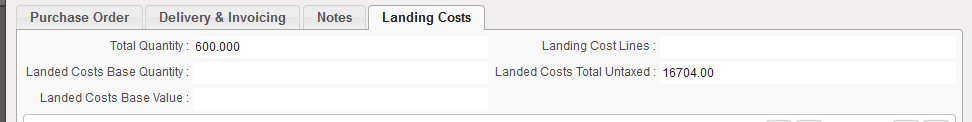
**Purchase order** merged [2 products]

There is no landed costs added therefore only the line amount appears, what is correct

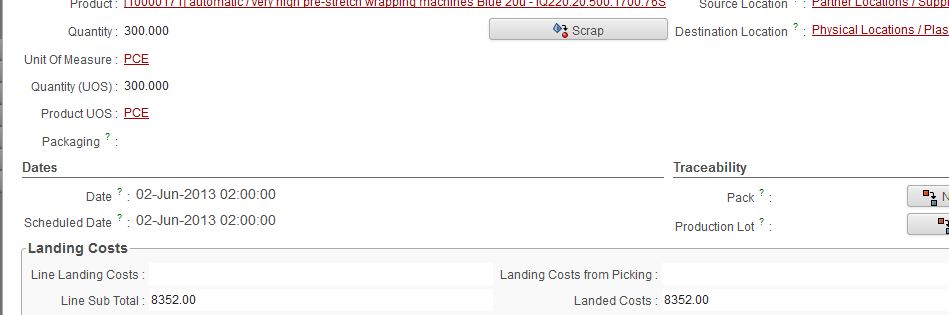




Total still correct

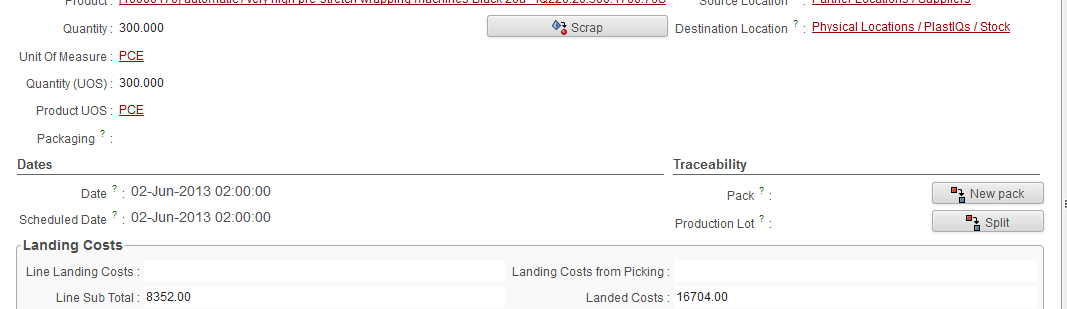


**SHIPPING**

Line 1 is correct: 

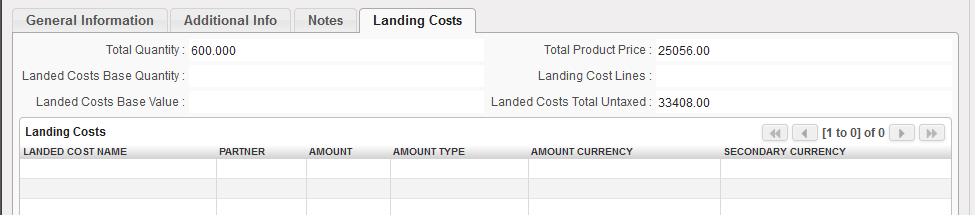
* Qty 300
* Subtotal 8352
* Landed costs (line) 8352

Line 2 is wrong:



* Qty 300
* Subtotal 8352
* Landed costs (line) **16704** should be in my opinion 8352 as it referes to the line

**Total shipping is wrong and calculates a wrong average price**



* Qty: 600 is correct
* Total product price is wrong: should be 16704
* Landed Costs Total Untaxed is wrong: should be 16704 [as for this example no additional costs were added for test purpose]

I am not (YET) a good python developer, so could you help me to correct the code?

**Python stock.py**

# -\*- coding: utf-8 -\*-

##############################################################################

#

# OpenERP, Open Source Management Solution

# Copyright (C) 2004-2010 Tiny SPRL (<http://tiny.be>).

# Copyright (C) 2010-2012 Camptocamp (<http://www.camptocamp.at>)

#

# This program is free software: you can redistribute it and/or modify

# it under the terms of the GNU Affero General Public License as

# published by the Free Software Foundation, either version 3 of the

# License, or (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU Affero General Public License for more details.

#

# You should have received a copy of the GNU Affero General Public License

# along with this program. If not, see <http://www.gnu.org/licenses/>.

#

##############################################################################

from osv import osv, fields

import decimal\_precision as dp

import logging

#----------------------------------------------------------

# Stock Move

#----------------------------------------------------------

class stock\_move(osv.osv):

\_inherit = "stock.move"

def \_landing\_cost(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs = 0.0

# landed costss for the line

for line in self.browse(cr, uid, ids):

if line.landed\_cost\_line\_ids:

for costs in line.landed\_cost\_line\_ids:

if costs.price\_type == 'value':

landed\_costs += costs.amount

else:

landed\_costs += costs.amount \* line.product\_qty

result[line.id] = landed\_costs

return result

def \_landing\_cost\_order(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs = 0.0

# landed costss for the line

for line in self.browse(cr, uid, ids):

# distrubution of landed costs of PO

if line.picking\_id.landed\_cost\_line\_ids:

if line.picking\_id.total\_amount and line.picking\_id.total\_amount > 0.0:

landed\_costs += line.picking\_id.landed\_cost\_base\_value / line.picking\_id.total\_amount \* line.price\_unit \* line.product\_qty

if line.picking\_id.quantity\_total and line.picking\_id.quantity\_total >0.0:

landed\_costs += line.picking\_id.landed\_cost\_base\_quantity / line.picking\_id.quantity\_total \* line.product\_qty

result[line.id] = landed\_costs

return result

def \_landed\_cost(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs = 0.0

# landed costss for the line

for line in self.browse(cr, uid, ids):

landed\_costs += line.product\_qty \* line.price\_unit

result[line.id] = landed\_costs

return result

def \_sub\_total(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

sub\_total = 0.0

for line in self.browse(cr, uid, ids):

sub\_total += line.product\_qty \* line.price\_unit\_net or 0.0

result[line.id] = sub\_total

return result

\_columns = {

'landed\_cost\_line\_ids': fields.one2many('landed.cost.position', 'move\_line\_id', 'Landed Costs Positions'),

'landing\_costs' : fields.function(\_landing\_cost, digits\_compute=dp.get\_precision('Account'), string='Line Landing Costs'),

'landing\_costs\_picking' : fields.function(\_landing\_cost\_order, digits\_compute=dp.get\_precision('Account'), string='Landing Costs from Picking'),

'landed\_cost' : fields.function(\_landed\_cost, digits\_compute=dp.get\_precision('Account'), string='Landed Costs'),

'sub\_total' : fields.function(\_sub\_total, digits\_compute=dp.get\_precision('Account'), string='Line Sub Total'),

'price\_unit\_net' : fields.float('Purchase Price', digits\_compute=dp.get\_precision('Account'), ),

}

stock\_move()

#----------------------------------------------------------

# Stock Picking

#----------------------------------------------------------

class stock\_picking(osv.osv):

\_inherit = "stock.picking"

def \_landed\_cost\_base\_value(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs\_base\_value = 0.0

for line in self.browse(cr, uid, ids):

if line.landed\_cost\_line\_ids:

for costs in line.landed\_cost\_line\_ids:

if costs.product\_id.landed\_cost\_type == 'value':

landed\_costs\_base\_value += costs.amount

result[line.id] = landed\_costs\_base\_value

return result

def \_landed\_cost\_base\_quantity(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs\_base\_quantity = 0.0

for line in self.browse(cr, uid, ids):

if line.landed\_cost\_line\_ids:

for costs in line.landed\_cost\_line\_ids:

if costs.product\_id.landed\_cost\_type == 'quantity':

landed\_costs\_base\_quantity += costs.amount

result[line.id] = landed\_costs\_base\_quantity

return result

def \_landed\_cost(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_costs = 0.0

# landed costss for the line

for line in self.browse(cr, uid, ids):

if line.move\_lines:

for ml in line.move\_lines:

landed\_costs += ml.landed\_cost

result[line.id] = landed\_costs

return result

def \_landing\_cost\_lines(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

landed\_cost\_lines = 0.0

for line in self.browse(cr, uid, ids):

if line.move\_lines:

for ml in line.move\_lines:

if ml.product\_qty > 0.0:

landed\_cost\_lines += ml.landing\_costs + ml.landing\_costs\_picking

result[line.id] = landed\_cost\_lines

return result

def \_quantity\_total(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

quantity\_total = 0.0

for line in self.browse(cr, uid, ids):

if line.move\_lines:

for ml in line.move\_lines:

if ml.product\_qty > 0.0:

quantity\_total += ml.product\_qty

result[line.id] = quantity\_total

return result

def \_amount\_total(self, cr, uid, ids, name, args, context):

if not ids : return {}

result = {}

amount\_total = 0.0

for line in self.browse(cr, uid, ids):

if line.move\_lines:

for ml in line.move\_lines:

if ml.product\_qty > 0.0 and ml.price\_unit:

amount\_total += ml.sub\_total

result[line.id] = amount\_total

return result

\_columns = {

'landed\_cost\_line\_ids': fields.one2many('landed.cost.position', 'picking\_id', 'Landed Costs Positions'),

'landed\_cost\_base\_value' : fields.function(\_landed\_cost\_base\_value, digits\_compute=dp.get\_precision('Account'), string='Landed Costs Base Value'),

'landed\_cost\_base\_quantity' : fields.function(\_landed\_cost\_base\_quantity, digits\_compute=dp.get\_precision('Account'), string='Landed Costs Base Quantity'),

'landing\_cost\_lines' : fields.function(\_landing\_cost\_lines, digits\_compute=dp.get\_precision('Account'), string='Landing Cost Lines'),

'landed\_cost' : fields.function(\_landed\_cost, digits\_compute=dp.get\_precision('Account'), string='Landed Costs Total Untaxed'),

'total\_amount' : fields.function(\_amount\_total, digits\_compute=dp.get\_precision('Account'), string='Total Product Price'),

'quantity\_total' : fields.function(\_quantity\_total, digits\_compute=dp.get\_precision('Product UoM'), string='Total Quantity'),

}

stock\_picking()

class stock\_partial\_picking(osv.osv\_memory):

\_inherit = "stock.partial.picking"

\_logger = logging.getLogger(\_\_name\_\_)

def \_product\_cost\_for\_average\_update(self, cr, uid, move):

res = super(stock\_partial\_picking, self).\_product\_cost\_for\_average\_update(cr, uid, move)

self.\_logger.debug('res stock\_partial\_picking `%s`', res)

res['cost'] = move.landed\_cost / move.product\_qty

self.\_logger.debug('res stock\_partial\_picking `%s`', res)

return res

stock\_partial\_picking()