Multi-company Guidelines

> **⚠ Warning**
> This tutorial requires good knowledge of Odoo. Please refer to the **basic tutorial (backend.html#howto-base)** first if needed.

As of version 13.0, a user can be logged in multiple companies at once. This allows the user to access information from multiple companies but also to create/edit records in a multi-company environment.
If not handled correctly, it may be the source of a lot of inconsistent multi-company behaviors. For instance, a user logged in both companies A and B could create a sales order in company A and add products belonging to company B to it. It is only when the user will log out from company B that access errors will occur for the sales order.
To correctly manage multi-company behaviors, Odoo's ORM provides multiple features:

- Company-dependent fields
- Multi-company consistency
- Default company
- Views
- Security rules

Company-dependent fields

When a record is available from multiple companies, we must expect that different values will be assigned to a given field depending on the company from which the value is set.
For the field of a same record to support several values, it must be defined with the attribute `company_dependent` set to `True`.

```
from odoo import api, fields, models

class Record(models.Model):
    _name = 'record.public'

    info = fields.Text()
    company_info = fields.Text(company_dependent=True)
    display_info = fields.Text(string='Infos', compute='_compute_display_info')

    @api.depends_context('company')
    def _compute_display_info(self):
        for record in self:
            record.display_info = record.info + record.company_info
```

> The `_compute_display_info` method is decorated with `depends_context('company')` (see `depends_context` (../reference/orm.html#odoo.api.depends_context)) to ensure that the computed field is recomputed depending on the current company (`self.env.company`).

When a company-dependent field is read, the current company is used to retrieve its value. In other words, if a user is logged in companies A and B with A as main company and creates a record for company B, the values of company-dependent fields will be that of company A.
To read the values of company-dependent fields set from another company than the current one, we need to ensure the company we are using is the correct one. This can be done with `with_company()` (../reference/orm.html#odoo.models.Model.with_company), which updates the current company.

```
# Accessed as main company (self.env.company)
val = record.company_dependent_field

# Accessed as desired company (company_B)
val = record.with_company(company_B).company_dependent_field
# record.with_company(company_B).env.company == company_B
```

> **⚠ Warning**
> Whenever you are computing/creating/... things that may behave differently in different companies, you should make sure whatever you are doing is done in the right company. It doesn't cost much to always use `with_company` to avoid problems later on.

```
@api.onchange('field_name')
def _onchange_field_name(self):
 self = self.with_company(self.company_id)
 ...

@api.depends('field_2')
def _compute_field_3(self):
 for record in self:
   record = record.with_company(record.company_id)
   ...
```

Multi-company consistency

When a record is made shareable between several companies by the mean of a `company_id` field, we must take care that it cannot be linked to the record of another company through a relational field. For instance, we do not want to have a sales order and its invoice belonging to different companies.
To ensure this multi-company consistency, you must:

- Set the class attribute `_check_company_auto` to `True` .
- Define relational fields with the attribute `check_company` set to `True` if their model has a `company_id` field.

On each `create()` (../reference/orm.html#odoo.models.Model.create) and `write()` (../reference/orm.html#odoo.models.Model.write), automatic checks will be triggered to ensure the multi-company consistency of the record.

```
from odoo import fields, models

class Record(models.Model):
    _name = 'record.shareable'
    _check_company_auto = True

    company_id = fields.Many2one('res.company')
    other_record_id = fields.Many2one('other.record', check_company=True)
```

> The field `company_id` must not be defined with `check_company=True`.

`Model._check_company(fnames=None)`
(https://github.com/odoo/odoo/blob/14.0/odoo/models.py#L3247)

Check the companies of the values of the given field names.

Parameters:

fnames (`list` (https://docs.python.org/3/library/stdtypes.html#list)) – names of relational fields to check

Raises:

UserError (../reference/orm.html#odoo.exceptions.UserError) – if the `company_id` of the value of any field is not in `[False, self.company_id]` (or `self` if `res_company`).

For `res_users` relational fields, verifies record company is in `company_ids` fields.

User with main company A, having access to company A and B, could be assigned or linked to records in company B.

> **⚠ Warning**
> The `check_company` feature performs a strict check ! It means that if a record has no `company_id` (i.e. the field is not required), it cannot be linked to a record whose `company_id` is set.

> When no domain is defined on the field and `check_company` is set to `True`, a default domain is added: `['|', '('company_id', '=', False), ('company_id', '=', company_id)]`

Default company

When the field `company_id` is made required on a model, a good practice is to set a default company. It eases the setup flow for the user or even guarantees its validity when the company is hidden from the view. Indeed, the company is usually hidden if the user does not have access to multiple companies (i.e. when the user does not have the group `base.group_multi_company`).

```
from odoo import api, fields, models

class Record(models.Model):
    _name = 'record.restricted'
    _check_company_auto = True

    company_id = fields.Many2one(
        'res.company', required=True, default=lambda self: self.env.company
    )
    other_record_id = fields.Many2one('other.record', check_company=True)
```

Views

As stated in above, the company is usually hidden from the view if the user does not have access to multiple companies. This is tested with the group `base.group_multi_company`.

```
<record model="ir.ui.view" id="record_form_view">
    <field name="name">record.restricted.form</field>
    <field name="model">record.restricted</field>
    <field name="arch" type="xml">
        <form>
            <sheet>
                <group>
                    <group>
                        <field name="company_id" groups="base.group_multi_company"/>
                        <field name="other_record_id"/>
                    </group>
                </group>
            </sheet>
        </form>
    </field>
</record>
```

Security rules

When working with records shared across companies or restricted to a single company, we must take care that a user does not have access to records belonging to other companies. This is achieved with security rules based on `company_ids`, which contains the current companies of the user (the companies the user checked in the multi-company widget).

```
<!-- Shareable Records -->
<record model="ir.rule" id="record_shared_company_rule">
    <field name="name">Shared Record: multi-company</field>
    <field name="model_id" ref="model_record_shared"/>
    <field name="global" eval="True"/>
    <field name="domain_force">
        ['|', ('company_id', '=', False), ('company_id', 'in', company_ids)]
    </field>
</record>


<!-- Company-restricted Records -->
<record model="ir.rule" id="record_restricted_company_rule">
    <field name="name">Restricted Record: multi-company</field>
    <field name="model_id" ref="model_record_restricted"/>
    <field name="global" eval="True"/>
    <field name="domain_force">
        [('company_id', 'in', company_ids)]
    </field>
</record>
```