Translating Modules

This section explains how to provide translation abilities to your module.

> If you want to contribute to the translation of Odoo itself, please refer to the **Odoo Wiki page (https://github.com/odoo/odoo/wiki/Translations)**.

Exporting translatable term

A number of terms in your modules are "implicitly translatable" as a result, even if you haven't done any specific work towards translation you can export your module's translatable terms and may find content to work with.

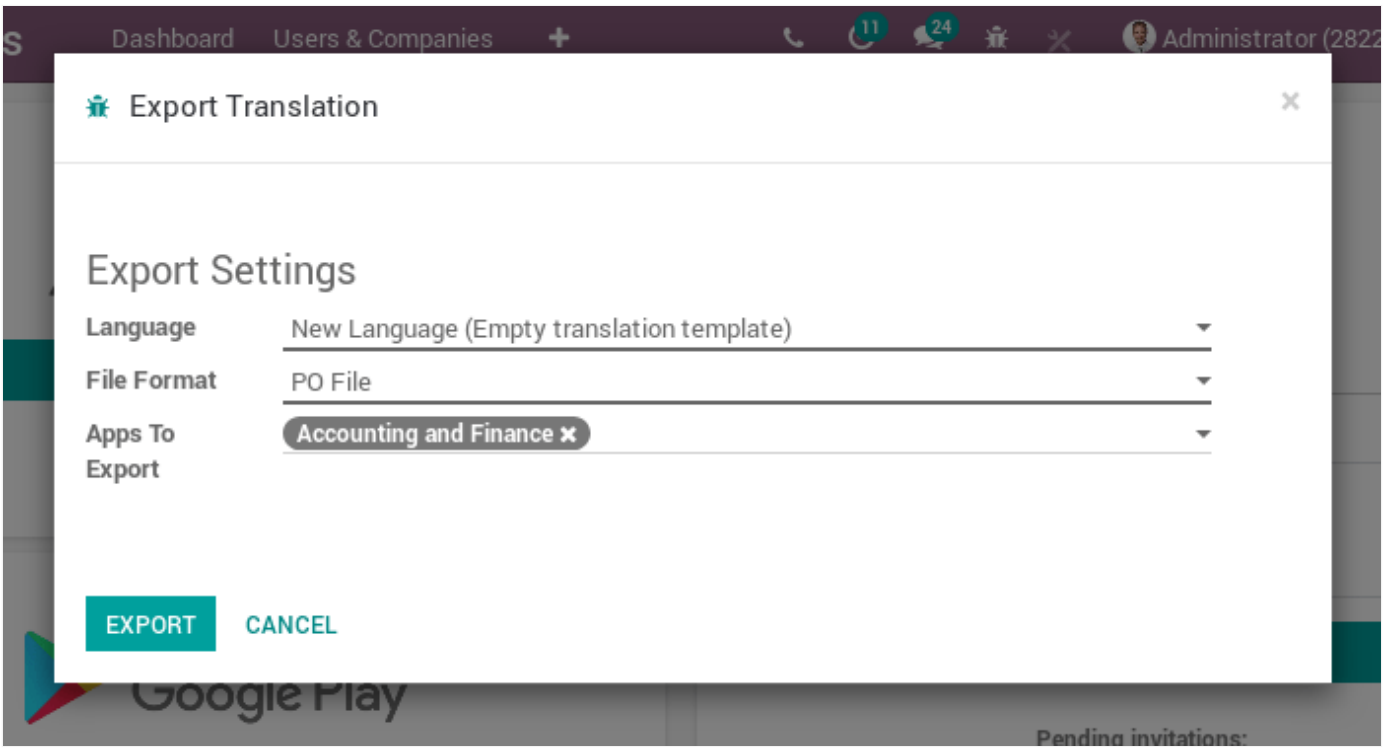
Translations export is performed via the administration interface by logging into the backend interface and opening **Settings ▸ Translations ▸ Import / Export ▸ Export Translations**

leave the language to the default (new language/empty template)

select the PO File (https://en.wikipedia.org/wiki/Gettext#Translating) format

select your module

click Export and download the file



(../_images/po-export.png)

This gives you a file called ***yourmodule.pot*** which should be moved to the ***yourmodule/i18n/*** directory. The file is a *PO Template* which simply lists translatable strings and from which actual translations (PO files) can be created. PO files can be created using msginit (https://www.gnu.org/software/gettext/manual/gettext.html#Creating), with a dedicated

translation tool like POEdit (https://poedit.net/) or by simply copying the template to a new file called `language.po`. Translation files should be put in `yourmodule/i18n/`, next to `yourmodule.pot`, and will be automatically loaded by Odoo when the corresponding language is installed (via Settings ▸ Translations ▸ Languages)

> translations for all loaded languages are also installed or updated when installing or updating a module

Implicit exports

Odoo automatically exports translatable strings from "data"-type content:

- in non-QWeb views, all text nodes are exported as well as the content of the `string`, `help`, `sum`, `confirm` and `placeholder` attributes
- QWeb templates (both server-side and client-side), all text nodes are exported except inside `t-translation="off"` blocks, the content of the `title`, `alt`, `label` and `placeholder` attributes are also exported
- for `Field` (orm.html#odoo.fields.Field), unless their model is marked with `_translate = False`:
  - their `string` and `help` attributes are exported
  - if `selection` is present and a list (or tuple), it's exported
  - if their `translate` attribute is set to `True`, all of their existing values (across all records) are exported
- help/error messages of `_constraints` and `_sql_constraints` are exported

Explicit exports

When it comes to more "imperative" situations in Python code or Javascript code, Odoo cannot automatically export translatable terms so they must be marked explicitly for export. This is done by wrapping a literal string in a function call.
In Python, the wrapping function is `odoo._()`:

```
title = _("Bank Accounts")
```

In JavaScript, the wrapping function is generally `odoo.web._t()`:

```
title = _t("Bank Accounts");
```

> **⚠ Warning**
> Only literal strings can be marked for exports, not expressions or variables. For situations where strings are formatted, this means the format string must be marked, not the formatted string

The lazy version of `_` and `_t` is `odoo._lt()` in python and `odoo.web._lt()` in javascript. The translation lookup is executed only at rendering and can be used to declare translatable properties in class methods of global variables.

Variables

Don't the extract may work but it will not translate the text correctly:

```
_("Scheduled meeting with %s" % invitee.name)
```

Do set the dynamic variables as a parameter of the translation lookup (this will fallback on source in case of missing placeholder in the translation):

```
_("Scheduled meeting with %s", invitee.name)
```

Blocks

Don't split your translation in several blocks or multiples lines:

```
# bad, trailing spaces, blocks out of context
_("You have ") + len(invoices) + _(" invoices waiting")
_t("You have ") + invoices.length + _t(" invoices waiting");

# bad, multiple small translations
_("Reference of the document that generated ") + \
_("this sales order request.")
```

Do keep in one block, giving the full context to translators:

```
# good, allow to change position of the number in the translation
_("You have %s invoices wainting") % len(invoices)
_.str.sprintf(_t("You have %s invoices wainting"), invoices.length);

# good, full sentence is understandable
_("Reference of the document that generated " + \
  "this sales order request.")
```

Plural

Don't pluralize terms the English-way:

```
msg = _("You have %(count)s invoice", count=invoice_count)
if invoice_count > 1:
  msg += _("s")
```

Do keep in mind every language has different plural forms:

```
if invoice_count > 1:
  msg = _("You have %(count)s invoices", count=invoice_count)
else:
  msg = _("You have one invoice")
```

Read vs Run Time

Don't invoke translation lookup at server launch:

```
ERROR_MESSAGE = {
  # bad, evaluated at server launch with no user language
  'access_error': _('Access Error'),
  'missing_error': _('Missing Record'),
}

class Record(models.Model):

  def _raise_error(self, code):
    raise UserError(ERROR_MESSAGE[code])
```

Don't invoke translation lookup when the javascript file is read:

```
# bad, js _t is evaluated too early
var core = require('web.core');
var _t = core._t;
var map_title = {
    access_error: _t('Access Error'),
    missing_error: _t('Missing Record'),
};
```

Do use lazy translation lookup method:

```
ERROR_MESSAGE = {
  'access_error': _lt('Access Error'),
  'missing_error': _lt('Missing Record'),
}

class Record(models.Model):

  def _raise_error(self, code):
    # translation lookup executed at error rendering
    raise UserError(ERROR_MESSAGE[code])
```

or **do** evaluate dynamically the translatable content:

```
# good, evaluated at run time
def _get_error_message(self):
  return {
    access_error: _('Access Error'),
    missing_error: _('Missing Record'),
  }
```

Do in the case where the translation lookup is done when the JS file is *read*, use `_lt` instead of `_t` to translate the term when it is *used*:

```
# good, js _lt is evaluated lazily
var core = require('web.core');
var _lt = core._lt;
var map_title = {
    access_error: _lt('Access Error'),
    missing_error: _lt('Missing Record'),
};
```