⚠ Warning
This tutorial requires knowledges about how to build a module in Odoo (see **Building a Module (../howtos/backend.html)**).

Building a localization module

When installing the `accounting` module, the localization module corresponding to the country code of the company is installed automatically. In case of no country code set or no localization module found, the `l10n_generic_coa` (US) localization module is installed by default.

For example, `l10n_be` will be installed if the company has `Belgium` as country.

This behavior is allowed by the presence of a *.xml* file containing the following code:

```
<function model="account.chart.template" name="try_loading_for_current_company">
   <value eval="[ref(module.template_xmlid)]"/>
</function>
```

Where `module.template_xmlid` is the **fully-qualified** xmlid of the corresponding template.

Usually located in the `data` folder, it must be loaded at the very last in the `__manifest__.py` file.

⚠ Danger
If the *.xml* file is missing, the right chart of accounts won't be loaded on time!

Configuring my own Chart of Accounts?

First of all, before I proceed, we need to talk about the templates. A template is a record that allows replica of itself. This mechanism is needed when working in multi-companies. For example, the creation of a new account is done using the `account.account.template` model. However, each company using this chart of accounts will be linked to a replica having `account.account` as model. So, the templates are never used directly by the company.

Then, when a chart of accounts needs to be installed, all templates dependent of this one will create a replica and link this newly generated record to the company's user. It means all such templates must be linked to the chart of accounts in some way. To do so, each one must reference the desired chart of accounts using the `chart_template_id` field. For this reason, we need to define an instance of the `account.chart.template` model before creating its templates.

```
<record id="..." model="account.chart.template">
    <!-- [Required] Specify the name to display for this CoA. -->
    <field name="name">...</field>

    <!-- [Required] Specify the currency. E.g. "base.USD". -->
    <field name="currency_id" ref="..."/>

    <!-- [Required] Specify a prefix of the bank accounts. -->
    <field name="bank_account_code_prefix">...</field>

    <!-- [Required] Specify a prefix of the cash accounts. -->
    <field name="cash_account_code_prefix">...</field>

    <!-- [Optional] Define a parent chart of accounts that will be installed just before
    <field name="parent_id" ref="..."/>

    <!-- [Optional] Define the number of digits of account codes. By default, the value i
    <field name="code_digits">...</field>

    <!-- [Optional] Boolean to show or not this CoA on the list. By default, the CoA is v
     This field is mostly used when this CoA has some children (see parent_id field). -->
    <field name="visible" eval="..."/>

    <!-- [Optional] Boolean to enable the Anglo-Saxon accounting. By default, this field
    <field name="use_anglo_saxon" eval="..."/>

    <!-- [Optional] Boolean to enable the complete set of taxes. By default, this field i
    This boolean helps you to choose if you want to propose to the user to encode the sal
    This last choice assumes that the set of tax defined on this template is complete. --
    <field name="complete_tax_set" eval="..."/>

    <!-- [Optional] Specify the spoken languages.
    /!\ This option is only available if your module depends of l10n_multilang.
    You must provide the language codes separated by ';', e.g. eval="'en_US;ar_EG;ar_SY'"
    <field name="spoken_languages" eval="..."/>
</record>
```

For example, let's take a look to the Belgium chart of accounts.

```
<record id="l10nbe_chart_template" model="account.chart.template">
    <field name="name">Belgian PCMN</field>
    <field name="currency_id" ref="base.EUR"/>
    <field name="bank_account_code_prefix">550</field>
    <field name="cash_account_code_prefix">570</field>
    <field name="spoken_languages" eval="'nl_BE'"/>
</record>
```

Now that the chart of accounts is created, we can focus on the creation of the templates. As said previously, each record must reference this record through the `chart_template_id` field. If not, the template will be ignored. The following sections show in details how to create these templates.

Adding a new account to my Chart of Accounts

It's time to create our accounts. It consists to creating records of **`account.account.template`** type. Each **`account.account.template`** is able to create an **`account.account`** for each company.

```
<record id="..." model="account.account.template">
    <!-- [Required] Specify the name to display for this account. -->
    <field name="name">...</field>

    <!-- [Required] Specify a code. -->
    <field name="code">...</field>

    <!-- [Required] Specify a type. -->
    <field name="user_type_id">...</field>

    <!-- [Required] Set the CoA owning this account. -->
    <field name="chart_template_id" ref="..."/>

    <!-- [Optional] Specify a secondary currency for each account.move.line linked to thi
    <field name="currency_id" ref="..."/>

    <!-- [Optional] Boolean to allow the user to reconcile entries in this account. True
    <field name="reconcile" eval="..."/>

    <!-- [Optional] Specify a group for this account. -->
    <field name="group_id" ref="...">

    <!-- [Optional] Specify some tags. -->
    <field name="tag_ids" eval="...">
</record>
```

Some of the described fields above deserve a bit more explanation.
The **`user_type_id`** field requires a value of type **`account.account.type`**. Although some additional types could be created in a localization module, we encourage the usage of the existing types in the account/data/data_account_type.xml (https://github.com/odoo/odoo/blob/14.0/addons/account/data/data_account_type.xml) file. The usage of these generic types ensures the generic reports working correctly in addition to those that you could create in your localization module.

⚠ Warning
Avoid the usage of liquidity **`account.account.type`**! Indeed, the bank & cash accounts are created directly at the installation of the localization module and then, are linked to an **`account.journal`**.

⚠ Warning
Only one account of type payable/receivable is enough.

Although the **`tag_ids`** field is optional, this one remains a very powerful feature. Indeed, this one allows you to define some tags for your accounts to spread them correctly on your reports. For example, suppose you want to create a financial report having multiple lines but you have no way to find a rule to dispatch the accounts according their **`code`** or **`name`**. The solution is the usage of tags, one for each report line, to spread and aggregate your accounts like you want.
Like any other record, a tag can be created with the following xml structure:

```
<record id="..." model="account.account.tag">
    <!-- [Required] Specify the name to display for this tag. -->
    <field name="name">...</field>

    <!-- [Optional] Define a scope for this applicability.
    The available keys are 'accounts' and 'taxes' but 'accounts' is the default value. --
    <field name="applicability">...</field>
</record>
```

As you can well imagine with the usage of tags, this feature can also be used with taxes.

An examples coming from the `l10n_be` module:

```
<record id="a4000" model="account.account.template">
    <field name="name">Clients</field>
    <field name="code">4000</field>
    <field name="user_type_id" ref="account.data_account_type_receivable"/>
    <field name="chart_template_id" ref="l10nbe_chart_template"/>
</record>
```

⚠ Warning

Don’t create too much accounts: 200-300 is enough.

Adding a new tax to my Chart of Accounts

To create a new tax record, you just need to follow the same process as the creation of accounts. The only difference being that you must use the `account.tax.template` model.

```
<record id="..." model="account.tax.template">
    <!-- [Required] Specify the name to display for this tax. -->
    <field name="name">...</field>

    <!-- [Required] Specify the amount.
    E.g. 7 with fixed amount_type means v + 7 if v is the amount on which the tax is appl
     If amount_type is 'percent', the tax amount is v * 0.07. -->
    <field name="amount" eval="..."/>

    <!-- [Required] Set the CoA owning this tax. -->
    <field name="chart_template_id" ref="..."/>

    <!-- [Required/Optional] Define an account if the tax is not a group of taxes. -->
    <field name="account_id" ref="..."/>

    <!-- [Required/Optional] Define an refund account if the tax is not a group of taxes.
    <field name="refund_account_id" ref="..."/>

    <!-- [Optional] Define the tax's type.
    'sale', 'purchase' or 'none' are the allowed values. 'sale' is the default value.
    Note: 'none' means a tax can't be used by itself, however it can still be used in a g
    <field name="type_tax_use">...</field>

    <!-- [Optional] Define the type of amount:
    'group' for a group of taxes, 'fixed' for a tax with a fixed amount or 'percent' for
    By default, the type of amount is percentage. -->
    <field name="amount_type" eval="..."/>

    <!-- [Optional] Define some children taxes.
    /!\ Should be used only with an amount_type with 'group' set. -->
    <field name="children_tax_ids" eval="..."/>

    <!-- [Optional] The sequence field is used to define order in which the tax lines are
    By default, sequence = 1. -->
    <field name="sequence" eval="..."/>

    <!-- [Optional] Specify a short text to be displayed on invoices.
    For example, a tax named "15% on Services" can have the following label on invoice "1
    <field name="description">...</field>

    <!-- [Optional] Boolean that indicates if the amount should be considered as included
    E.g. Suppose v = 132 and a tax amount of 20.
    If price_include = False, the computed amount will be 132 * 0.2 = 26.4.
    If price_include = True, the computed amount will be 132 - (132 / 1.2) = 132 - 110 =
    <field name="price_include" eval="..."/>

    <!-- [Optional] Boolean to set to include the amount to the base. False by default.
     If True, the subsequent taxes will be computed based on the base tax amount plus the
     E.g. suppose v = 100, t1, a tax of 10% and another tax t2 with 20%.
     If t1 doesn't affects the base,
     t1 amount = 100 * 0.1 = 10 and t2 amount = 100 * 0.2 = 20.
     If t1 affects the base,
     t1 amount = 100 * 0.1 = 10 and t2 amount = 110 * 0.2 = 22.  -->
    <field name="include_base_amount" eval="..."/>

    <!-- [Optional] Boolean false by default.
     If set, the amount computed by this tax will be assigned to the same analytic accoun
    <field name="analytic" eval="..."/>
    <!-- [Optional] Specify some tags.
    These tags must have 'taxes' as applicability.
    See the previous section for more details. -->
    <field name="tag_ids" eval="...">

    <!-- [Optional] Define a tax group used to display the sums of taxes in the invoices.
    <field name="tax_group_id" ref="..."/>

    <!-- [Optional] Define the tax exigibility either based on invoice ('on_invoice' valu
    either based on payment using the 'on_payment' key.
    The default value is 'on_invoice'. -->
    <field name="tax_exigibility">...</field>

    <!-- [Optional] Define a cash basis account in case of tax exigibility 'on_payment'.
    <field name="cash_basis_account" red="..."/>
</record>
```

An example found in the `l10n_pl` module:

```
<record id="vp_leas_sale" model="account.tax.template">
    <field name="chart_template_id" ref="pl_chart_template"/>
    <field name="name">VAT - leasing pojazdu(sale)</field>
    <field name="description">VLP</field>
    <field name="amount">1.00</field>
    <field name="sequence" eval="1"/>
    <field name="amount_type">group</field>
    <field name="type_tax_use">sale</field>
    <field name="children_tax_ids" eval="[(6, 0, [ref('vp_leas_sale_1'), ref('vp_leas_sal
    <field name="tag_ids" eval="[(6,0,[ref('l10n_pl.tag_pl_21')])]"/>
    <field name="tax_group_id" ref="tax_group_vat_23"/>
</record>
```

Adding a new fiscal position to my Chart of Accounts

> If you need more information about what is a fiscal position and how it works in Odoo, please refer to **How to adapt taxes to my customer status or localization (https://www.odoo.com/documentation/user/online/accounting/others/taxes/application.html)**.

To create a new fiscal position, simply use the `account.fiscal.position.template` model:

```
<record id="..." model="account.fiscal.position.template">
    <!-- [Required] Specify the name to display for this fiscal position. -->
    <field name="name">...</field>

    <!-- [Required] Set the CoA owning this fiscal position. -->
    <field name="chart_template_id" ref="..."/>

    <!-- [Optional] Add some additional notes. -->
    <field name="note">...</field>
</record>
```

Adding the properties to my Chart of Accounts

When the whole accounts are generated, you have the possibility to override the newly generated chart of accounts by adding some properties that correspond to default accounts used in certain situations. This must be done after the creation of accounts before each one must be linked to the chart of accounts.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <record id="l10n_xx_chart_template" model="account.chart.template">

        <!-- Define receivable/payable accounts. -->
        <field name="property_account_receivable_id" ref="..."/>
        <field name="property_account_payable_id" ref="..."/>

        <!-- Define categories of expense/income account. -->
        <field name="property_account_expense_categ_id" ref="..."/>
        <field name="property_account_income_categ_id" ref="..."/>

        <!-- Define input/output accounts for stock valuation. -->
        <field name="property_stock_account_input_categ_id" ref="..."/>
        <field name="property_stock_account_output_categ_id" ref="..."/>

        <!-- Define an account template for stock valuation. -->
        <field name="property_stock_valuation_account_id" ref="..."/>

        <!-- Define loss/gain exchange rate accounts. -->
        <field name="expense_currency_exchange_account_id" ref="..."/>
        <field name="income_currency_exchange_account_id" ref="..."/>

        <!-- Define a transfer account. -->
        <field name="transfer_account_id" ref="..."/>
    </record>
</odoo>
```

For example, let's come back to the Belgium PCMN. This chart of accounts is override in this way to add some properties.

```xml
<record id="l10nbe_chart_template" model="account.chart.template">
    <field name="property_account_receivable_id" ref="a4000"/>
    <field name="property_account_payable_id" ref="a440"/>
    <field name="property_account_expense_categ_id" ref="a600"/>
    <field name="property_account_income_categ_id" ref="a7010"/>
    <field name="expense_currency_exchange_account_id" ref="a654"/>
    <field name="income_currency_exchange_account_id" ref="a754"/>
    <field name="transfer_account_id" ref="trans"/>
</record>
```

How to create a new bank operation model?

How a bank operation model works exactly in Odoo? See **Configure model of entries (https://www.odoo.com/documentation/user/online/accounting/bank/reconciliation/configure.html)**.

Since `V10`, a new feature is available in the bank statement reconciliation widget: the bank operation model. This allows the user to pre-fill some accounting entries with a single click. The creation of an `account.reconcile.model.template` record is quite easy:

```
 <record id="..." model="account.reconcile.model.template">
     <!-- [Required] Specify the name to display. -->
     <field name="name">...</field>

     <!-- [Required] Set the CoA owning this. -->
     <field name="chart_template_id" ref="..."/>

     <!-- [Optional] Set a sequence number defining the order in which it will be display
     By default, the sequence is 10. -->
     <field name="sequence" eval="..."/>

     <!-- [Optional] Define an account. -->
     <field name="account_id" ref="..."/>

     <!-- [Optional] Define a label to be added to the journal item. -->
     <field name="label">...</field>

     <!-- [Optional] Define the type of amount_type, either 'fixed' or 'percentage'.
     The last one is the default value. -->
     <field name="amount_type">...</field>

     <!-- [Optional] Define the balance amount on which this model will be applied to (10
     Fixed amount will count as a debit if it is negative, as a credit if it is positive.
     <field name="amount">...</field>

     <!-- [Optional] Define eventually a tax. -->
     <field name="tax_id" ref="..."/>

     <!-- [Optional] The sames fields are available twice.
     To enable a second journal line, you can set this field to true and
     fill the fields accordingly. -->
     <field name="has_second_line" eval="..."/>
     <field name="second_account_id" ref="..."/>
     <field name="second_label">...</field>
     <field name="second_amount_type">...</field>
     <field name="second_amount">...</field>
     <field name="second_tax_id" ref="..."/>
</record>
```

How to create a new dynamic report?

If you need to add some reports on your localization, you need to create a new module named **l10n_xx_reports**. Furthermore, this additional module must be present in the `enterprise` repository and must have at least two dependencies, one to bring all the stuff for your localization module and one more, `account_reports`, to design dynamic reports.

```
'depends': ['l10n_xx', 'account_reports'],
```

Once it’s done, you can start the creation of your report statements. The documentation is available in the following slides (https://www.odoo.com/slides/slide/how-to-create-custom-accounting-report-415).