Odoo Accounting v9. Draft.

Table of Content

Introduction

Odoo Accounting is as good as accounting modules of traditional ERPs. But it does not have the quality of dedicated accounting software. Our reference, in terms of usability (not features), is Xero.com → Odoo v9 must have a better usability than Xero.

Odoo Accounting's Unique Selling Proposition will be:

- Super smooth integration with banks:
    - Download / Import bank statements
    - Smart reconciliation with Invoices
    - Manage payments easily
    - Upload payments / send checks
- Strong usability:
    - super clear & easy to use, no complex config or menus
    - fast to perform daily operations.

This project covers features like assets and expenses. We will work in three phases:

- Phase 1: Generic improvements: Oct 2014 → Feb 2015
- Phase 2: Localisation of key countries: Feb 2015 → May 2015
- Phase 3: Put in production and fine tune: May 2015 → July 2015
- Phase 4: Improve extra features like budget, analytic.

- extra questions to address in workshops
    - http://pad.odoo.com/p/accounting_roadmap_v9_extra_points
    - how can we improve payment terms with discounts and/or extra? (check with arthur)
    - to be tested: fiscal position on the SO based on the delivery address instead of the invoice address?
    - outstanding receivable: currencies revaluation at the end of FY/period

Phase 1

Model Changes

Cash vs Accrual Accounting (DONE)

Currently, Odoo Accounting is using the accrual accounting method but in some countries (e.g US), the norm is merely to do cash accounting for some businesses (usually services). More details on the difference between both method in:
http://www.irs.gov/publications/p538/ar02.html#en_US_201212_publink1000270633

Every report can have an option to generate the report using a cash-based method or an accrual based one (on the Tax Summary report for example, this will allow us to have a VAT on payment reporting). This option will be selected by default using this information from the company.

The reports filters the journal items to use according to the method:

- Accrual: the current method: we use all journal items in the reports
- Cash:
    - if journal entries are not linked to a “receivable” or “payable” account, they are used in the report
    - if journal entries are linked to either a receivable or a payable account: if these journal items are not reconciled, they are not used in the report

Note that partial reconciliation have also an impact in cash-based reports (checked in Xero), not only full reconciliation are shown in cash-based reports. That means that if a sale of 100€ with 10€ of vat is made, when receiving a payment of 11€, we recognize 10% of the sale and 10% of the vat too. Because of this, we gonna need additional stored field to know the debit and credit to use in cash-based reports.

TODO:

- add a field on res.company to choose between cash or accrual method (fields.selection, widget=radio, default=’accrual’)
- add the following fields on account.move.line:
    - debit_cash_basis: float
    - credit_cash_basis: float
    - at the creation of account.move.line, if the journal entry is under a sale/purchase journal: those fields are set to 0. Otherwise debit_cash_basis = debit and credit_cash_basis = credit

- at the reconciliation, on the reconciled items under a sale/purchase journal update debit_cash_basis and credit_cash_basis fields at the prorata of the reconciled amount on the total sold/purchased
- in reports where this option has a meaning (P&L, BS, Tax Summary...), add the option “cash basis ”. If it is checked, use the new columns instead of the classic debit credit ones.

Review Reconciliation (DONE)

Terminology change (DONE)

The system will be entirely reviewed in order to clear the confusion about the term “reconciliation” which, in real-english, doesn’t refer to the behavior we have in Odoo. We need to distinguish 2 use cases that currently use “reconciliation” as terminology (although even in french there exists 2 different words for that):

1. I want to make the link between one/several existing invoice(s) and one/several existing payment(s). Here we should speak about “matching” the payments and the invoices. In french we can continue talking about “réconcilier des factures et paiements”.
2. I want to make the link between a bank statement I imported and existing journal entries that are already present in my system. Here we should speak about the “reconciliation” of the bank statement. In french we should talk about “rapprocher des extraits bancaires”.

Because of the unordered way we created the document, you’ll still find in it references to both use cases using “reconciliation” as a single terminology. Please don’t be confused and replace mentally with “matching” everywhere bank statements aren’t involved.

TODO:

- Rename “reconciliation” with matching

Full vs Partial matching (DONE)

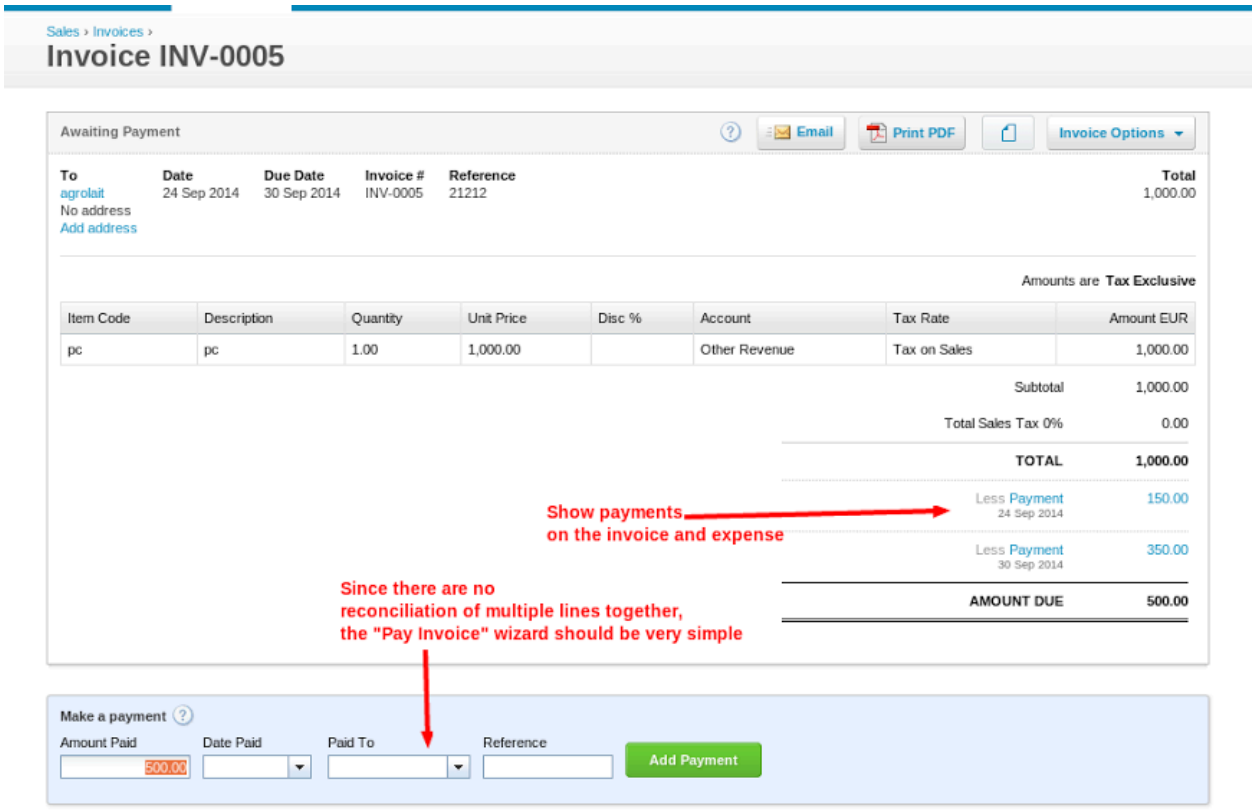
Current:

- Reconciliation concept is ok.
- Partial reconciliation is not good. → When we **partially reconcile** 2 payments with 3 invoices, it’s impossible to know the due amount for each invoice.

We propose to simplify partial reconciliation. Instead of using the same concept as full reconcile (link several lines together), a partial reconciliation is a direct link from one journal item to another: you can link the journal item of a payment to one and only one receivable journal item (e.g. from an invoice).

TODO:

- reconcile_partial_id field:
    - remove: fields.many2one('account.move.reconcile')
    - add: fields.many2one('account.move.line')
- Review the due amount computed field on invoices and expenses to reduce partial payments:
    - Unreconciled receivables/payable journal items - partial payments
- Show payments done on the invoice and expenses
- Simplify “Pay Invoice” wizard:
    - no more selection of debit/credit lines (the 2 one2many fields)
    - should not be an account.voucher anymore, create a dedicated object for payments.
- Remove the code that tries to merge several partial reconciliation into a bigger one.



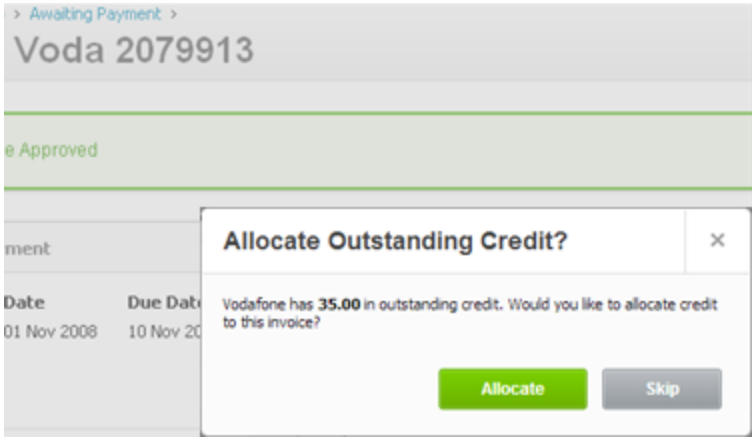
With the new bank statement that allows to reconcile on the fly, a user may never have to do a reconciliation[1]:

1

https://www.odoo.com/forum/help-1/question/how-can-i-reconcile-general-accounts-from-my-bank-statement-56566

- When importing bank statements, the bank reconciliation process do the reconciliation of the payment with the invoices
- When creating invoices, if there is an outstanding payment for this customer, the system proposes to assign it to the invoice.

TODO: When confirming an invoice, if the customer has an outstanding credit, the system proposes to automatically assign it to this invoice as shown by the screenshot below:



Of course, the reconciliation process still exists but it should not be a mandatory operation anymore. Users will not have to worry anymore about the “reconciliation” process. (which is not a simple concept to explain)

Nearly all journal items will be reconciled at the bank statement reconciliation process[2] (payment after invoice) or at the invoice validation (payment before invoice).

Taxes

Information on Journal Items (DONE)

In current versions of Odoo, a supplier invoice/refund may generate the following journal items (based on Belgium accounting, a purchase refund invoice with 21% of VAT on a single product costing 252.88):

Journal Entry	Account	Taxes	Debit	Credit	Tax Code	Tax Amount
1	440 Payable		305.98	0		

[2] The bank statement reconciliation (a new concept in Odoo v9) is not the same thing than the journal item reconciliation. (a concept already existing in Odoo)

1	411 VAT		0	53.10	63	53.10
1	611 Charges		0	0	85	252.88
1	611 Charges	21%	0	252.88	82	-252.88

The main problem we want to solve here is the third line with debit=credit=0 which is a technical hack to have a line with a Tax Code 85 and Tax amount, but not accounting entries. (we often have accountants that reports this as a bug)

We think we should split:

- The data about the journal items
- The Tax Legal Statement that should be a report like the P&L and BS. We don't need to store the tax codes in the journal items, we just need to know which journal items have been created by which tax line.

The table below are the journal items for the same transaction than the one described above.

Journal Entry	Account	Taxes	Debit	Credit	Tax Line ID
1	440 Payable		305.98	0	
1	411 VAT		0	53.10	1
1	611 Charges		0	0	2
1	611 Charges	21%	0	252.88	

Instead of having a tax codes referred in the journal items to compute the base and tax amounts for the tax statement, the Tax Statement Reports will compute these information using journal items linked to specific taxes.

The main advantages are:

- No redundancy between debit/credit and Tax Amount → no possible errors
- No more strange journal items with debit=credit=0
- We remove the concept of tax code structure that was complex to understand
- The tax statement becomes a normal reports like the P&L / BS → same report engine, less code.

Tax creation (DONE)

A tax has several tax lines but cannot be defined as a group of taxes. Tax groups can be implemented using several tax lines.

Every document that refers to a tax (invoices, expenses, …) should use a many2many field for taxes instead of many2one. (e.g. Sales Receipt and Purchase Receipt should be changed).

This way, Sales Receipt and Purchase Receipts are multi-taxes. The reason why we don't want tax groups anymore are:

- because it's a bit overlapping with the one2many tax lines
- because we need to be able to make the tax declaration report, and it makes things more complicated if we allow tax groups. (e.g. is a tax group a tax in itself or just a shortcut to select two taxes?)

That means that some of the fields that were previously on the tax are moved at the line level: include in base amount, python code.

The tax lines are simplified to the minimum:

- a (hidden) sequence (to make it draggable) ,
- a computation type (available types for the tax part are percent, fixed, python and we will add division which is needed in some countries like brazil and Bolivia),
- the general account (not required - if left empty the journal item created by this tax line will use the account of the sale / base.),
- the amount,
- a tax type that can be used to group taxes in reports, e.g.
    - Deductible / Non Deductible taxes
- the python code (moved into another extra module)
- the "include in base amount" field

Note on how to compute percent and division taxes:

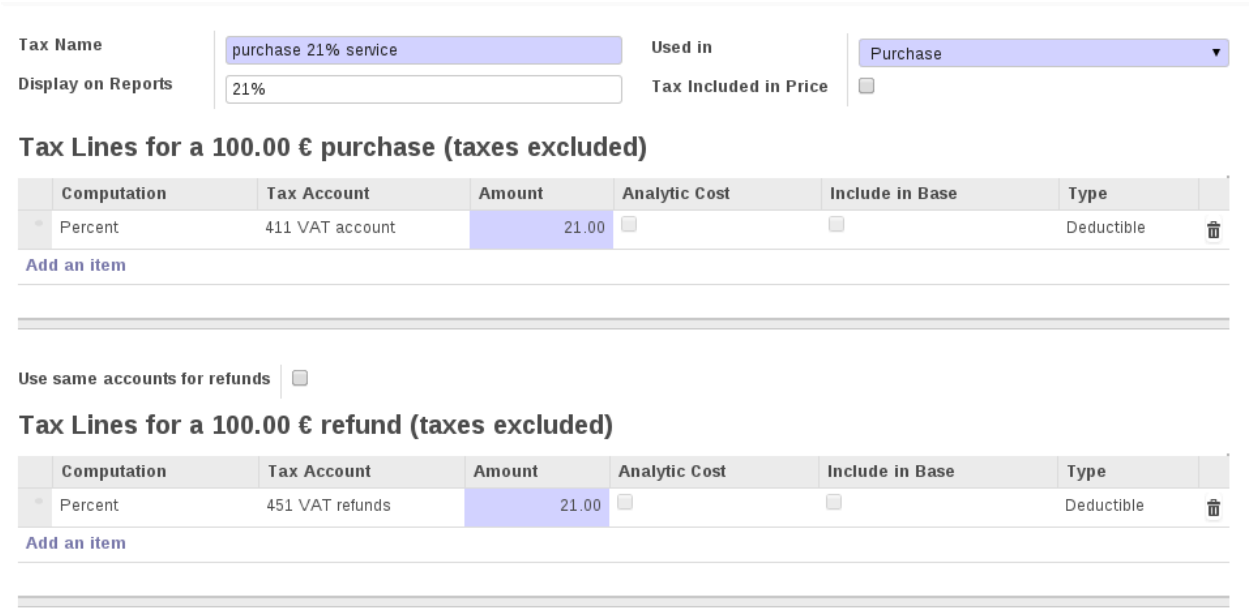
- Percent: (current mechanism of Odoo)
    - Price with Tax Excluded: Tax = Price * 21%
    - Price with Tax Included: Tax = Price - (Price / (100% + 21%))
- Division: (a new tax type)
    - Price with Tax Excluded: Tax = Price / (100% - 12%) - Price
    - Price with Tax Included: Tax = Price * 12%

Note:

- we should use a better name than division: "Percentage of Price Tax Excluded" and "Percentage of Price Tax Included".

- we also add a checkbox to avoid having to define twice the same tax lines for invoices and refunds if there's no difference in between.

The belgian purchase tax of 21% thus becomes as simple to define as this:



Tax Statement (DONE)

In current Odoo's version, the tax statement report is based on tax codes. There was a chart of tax code that allows localization to create a tax statement report based on the sum of the field tax_amount from account.move.line, grouped by tax_code_id.

We think it's an issue because:

- most users did not understand the tax code concept
- some localization computes the tax statement based on accounts, not on tax codes

In v9, those information will be computed by the reports. There will be a generic report called "Belgium Tax Statement" that will just print the summary of base/tax amounts per tax. Moreover, each localization will have the possibility to define a report that gives the exact same information as previously the tax codes were providing.

Let's take back our example of a belgian supplier refund on a single product costing 252.88 with 21% of VAT, and see how both reports can be built. As said before, we had previously (v8) the following journal items:

Journal Entry	Account	Debit	Credit	Tax_code_id	Tax Amount

1	440 Payable	305.98	0		
1	411 VAT	0	53.10	63	53.10
1	611 Charges	0	0	85	252.88
1	611 Charges	0	252.88	82	-252.88

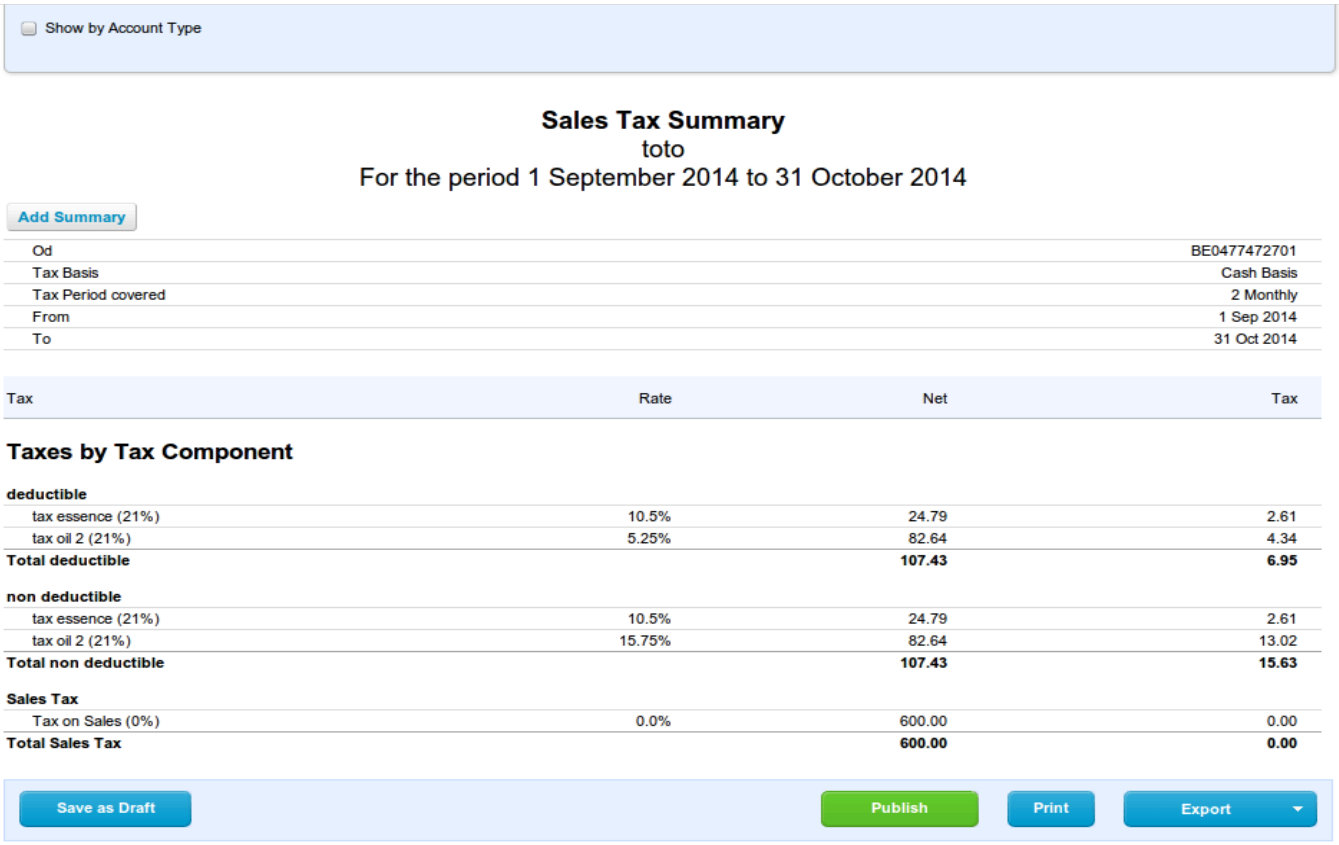
While, in the v9, given the tax defined as explained in the above section, we will have this instead:

Journal Entry	Account	Debit	Credit	Tax_ids	Tax_line_id
1	440 Payable	305.98	0		
1	411 VAT	0	53.10		#42 (1st line of purchase 21% service)
1	611 Charges	0	252.88	#35 (purchase 21% service)	

Note that:

- The tax_ids is a new many2many field that will contain all the taxes involved in the operation.
- The tax_line_id is a new many2one field that will held the reference to the tax line that created the journal item.

The generic tax summary (DONE)



Show by Account Type

Sales Tax Summary
toto
For the period 1 September 2014 to 31 October 2014

Add Summary

Od	BE0477472701
Tax Basis	Cash Basis
Tax Period covered	2 Monthly
From	1 Sep 2014
To	31 Oct 2014

Tax	Rate	Net	Tax
Taxes by Tax Component			
deductible			
tax essence (21%)	10.5%	24.79	2.61
tax oil 2 (21%)	5.25%	82.64	4.34
Total deductible		**107.43**	**6.95**
non deductible			
tax essence (21%)	10.5%	24.79	2.61
tax oil 2 (21%)	15.75%	82.64	13.02
Total non deductible		**107.43**	**15.63**
Sales Tax			
Tax on Sales (0%)	0.0%	600.00	0.00
Total Sales Tax		**600.00**	**0.00**

Save as Draft Publish Print Export

That report just loops on each tax line and for each gives the Net amount (sum of journal items having the related tax in the tax_ids column) and the Tax amount (sum of journal items having that tax line in tax_line_id. The tax lines are grouped by tag (deductible, non deductible) and the 2 first columns simply show its related tax name and short name.

Handling the tax declaration in localization modules (DONE)

Because we need to be able to have the tax declaration in a legal format, the tax summary report isn't enough. That's the reason why we want the localization modules to be able to define reports that will gives the exact same information as the tax codes were providing in v8.
For that, we think that localization modules should define account.financial.report that will have each previously existing tax code as a line in the report and that will compute the amounts for them.

For example, in the belgian localization module we will define the following formulae that will give the exact same information we had before on the account.move.line.

tax code	formula

63	sum of journal items having #42 in tax_line_id
82	sum of debit of journal items having #35 in tax_ids
85	sum of journal items having #35 in tax_ids to check

More explanation on the account.financial.report and the possibilities offered by it in a later section.

TODO:

- remove account.tax.code object
- create a account.tax.group object:
    - Name field, translate = True
    - As Data:
        - Deductible
        - Non Deductible
- implement the new tax system
- implement the generic tax statement report
- put the “Python” computation on tax in a separate module
- implement the belgian tax statement as an example of complex statement

Tax retention and withdrawing taxes (DONE)

We use a new concept of Tax Group to group txes the way you want: Eco Taces, Retention, VAT.

Periods & Fiscal Years management (DONE)

The concept of period is mostly used for:

- being able to close a period (to avoid posting journal entries when you already reported to the estate, for example).
- Set an accounting period on documents

The “period” concept has a few issues:

- Usability: creation of fiscal years and the related periods. We have seen a lot of customers locked in January 1st. They could not invoice because they forgot to create periods. (and they forgot how to do it)
- Special Period for Opening Entries: makes comparisons of periods very complex (e.g. one column per quarter, how do you compare Q4 2014 and Q1 2015: most users don’t know how to do that in v8)

- Users often do the mistake to filter on dates rather than periods. (for instance the get the turnover of the last month).
- When working with multiple companies, it's confusing to have several periods for the same time. It's a mess to use and may lead to strange behaviour if you select the wrong one.
- simplify the reports options where we can just filter on dates. Delegate the grouping (month, quarter, year) at the report level, so that it's more powerful.

Instead of using an object and choose the period with a many2one on documents, we will replace the period by an "Accounting Valuation Date" on documents. For example, there will be two dates on the invoice:

- the Invoice Date and the Accounting Date.
- the accounting date will be set automatically when "on change" the invoice date
- The accounting date field is:
    - hidden on customer invoices
    - visible on supplier invoices

In terms of usability, it's even better because some documents don't need two dates (or a date and a period). As an example, on a customer invoice or a misc. operations, you can not have a date AND a period. Only one date should be set. (it's illegal in most countries to have two different values) So, this new system is more accurate. (the only use case when you may have a different document date and a valuation date is for documents sent by third parties, like vendor bills on bank statements)

We also need to take care of the period closing. We plan to have a freeze date on fiscal years, that will prevent creating/modifying journal entries in a journal if its accounting date is before the freeze date of the journal. We actually want two mechanism: one for accountant, one for advisors. That way, the advisor can lock the accountant while he is doing the fiscal year closing.

The date grouping/filtering will be improved on reports to easily: report on a month, quarter, year, compare quarters, … That logic is much more advanced than period that were limited to report/group by months.

Fiscal years is only needed in reports:

- Computation of the opening balance according to the account type
- Easy selection of the filters (This Year, Previous Year) and comparisons with last year.

So, the fiscal year can be replaced by a field on the company, telling what's the first day of every fiscal year, like "January 1". It's not a date field, it's valid for all years.

The first year of a company incorporation, you may have a fiscal year of 15 months. That's ok as we can select any date range when doing a report. You will have to select custom dates and choose these 15 months when you print a report. It works just fine.

Reports are improved to easily compare across months/quarters/years. That way, you don't need the period concept to group elements, you can delegate it to the report engine that is way more powerful.

TODO:

- replace the period_id with a new date_account field
- remove account.period table
- remove account.fiscal_year table
- in creation of the analytic lines, use this new accounting date
- On the company, add a freeze date: freeze all journal items until __/__/____ (will replace the concept of closing a period).
- On the company, add two fields:
    - fiscalyear_day: integer (1-31)
    - fiscalyear_month: selection of the month (1-12)
- we need to improve the filtering on dates ranges, for keeping easiness on report options: we should be able to filter in one click on periods or dates: Last quarter, Year to Date (using Fiscal Year 1st day), This Month, … (should be ok with the task of Gery)
- Remove the field on the journal: force the Invoice Date to be in the related period. → this will be managed by document, not by journals (e.g. supplier invoice: 2 dates, customer invoices: 1 date, expenses: 1 date on document, 1 date per line, ...)

All operations will be based on dates, not on fiscal years anymore.

Fiscal Year Closing (DONE)

We plan to remove the concept of opening entries when closing a fiscal year:

- Reports are always correct, even if the preceding fiscal year has not been closed
- Easier for end user: no need to constantly generate opening entries for reporting, remove them, …
- No redundancy on receivable/payable entries
- No need for an "opening" period
- No need for an "opening" journal

Every report will implement correctly the "Opening Balance", at the report level, not depending on opening entries journal items. So, for the accountant, this does not change anything on reports. He just do not have to worry about generating opening entries. --> The opening

balance is computed at the reporting level, not by creating Journal Entries (it's already partially like that in some reports)

The “Opening Balance” of an account should be computed according to the account type (P&L or BS) and this should not be diff, Indeed, it should be the role of the report to compute the initial balance whatever the opening entries have been created or not.

The biggest advantages would be to have always the reports that are correct without any work from the accountants. No need anymore to re-generate each time the opening entries before printing the balance sheet. No need to have a crappy special move where everything is reconciled together.

In the balance sheet, we add a line “Current Year Earnings” with the non affected result, that is not an account, but computed based on the P&L.

As a result, closing a fiscal year is only two operations:

- creating the openeing journal entry, only two journal items. (e.g. Credit: Current Year Earnings, Debit: Retained Earnings)
- freezing the period/fy by changing the lock date on the company

Thus, we will remove the wizard as it’s just one journal entry (that has only two lines) to close a fiscal year.

Setting Initial Account Balances (Cancelled)

We need a tool to set the account balances for companies that start:

- In Settings / Accounting, add a button: Set Initial Balances (put this in the planner too)
- This opens an editable list view of all accounts with debit and credit being read/write
- If you write in debit/credit, it create a journal item in a “Centralised” and “Miscellaneous” journal with the difference between the current debit and the given one.

This approach works for both companies that start from nothing or those having used Odoo for billing during a few months and that deploy accounting after.

NOTE: we will not implement this! → Too complex for a problem that does not exists. (what’s the journal, what’s the counterpart account of an entry, …)

Sequences management (DONE, to fix)

We should be able to easily manage the sequences. Currently the usability is a pain. Actually, people shouldn't at all be aware of "sequences".

Since the Fiscal year are not required anymore in order for the system to work (we can use them but we can also simply use a set of date range), we can't base ourselves on the FY for the sequence.

What we want is the ability to create sequence by range of date, and that feature could also be used in other module and not only accounting.

Therefore, we will change directly the ir.sequence object. What needs to be done is the following:

- add a boolean fields on the ir.sequence object in order to know if we should use the date range fields or not
- add a one2many field 'date_range_ids' that points to a new object ir.sequence.date_range
    - that new object only has a few fields:
        - range_from
        - range_to
        - number_next
        - number_next_display (fields.function that does the same as the number_next_actual field in ir.sequence) used to display the information on the view and let the user change it if he wants
    - by default the date range will be all the year (from 1st january till 31th december)
- in the get_next method called to get the next incremental number, add a condition to check if we use date range or not.
  if self.use_date_range:
    if context.get(date,today()) exists in self.date_range_ids:
      increment number in correct ir.sequence.date_range and return that one
    else: //we don't have a ir.sequence.date_range yet
      create new ir.sequence.date_range and start numbering at one
- add a view to see the ir.sequence.date_range

Note: when creating a new ir.sequence.date_range, check if we don't already have a ir.sequence.date_range that exist for the current year, if yes, take the one with the closest range_to (in the past), increment it by one day and use that as range_from for the new ir.sequence.date_range (range_to goes by default to end of the year, except if we have a ir.sequence.date_range object with a range_from in the future in which case we should use that day minus one as range_to)

Accounts (DONE)

Account Hierarchies (DONE)

Remove the hierarchies in the chart of accounts and replace it by **optional** tags (that can be used to create hierarchies in the CoA report → keep in mind that the CoA will be a report on the screen, not anymore a tree view. This will allow smart filters, options like comparisons, ...):

- The concept of “Parent Account” is too complex. We often see accountants that creates accounts without parents.
- The concept of “View Accounts” or “Root Node” is too complex
- Being forced to create the whole structure in order to start is too complex too. (note from Ronda: for the U.S. version, consider starting with the accounts listed on the 1120 tax form schedule L, page 6 of PDF: U.S. chart of account.)

What will it solve:

- Easier creation of a new Accounts (no parent, no view accounts, no structure to pre-create)
- Easier to import the initial chart of account (just a CSV of account names and codes)
- Working on a list of accounts is much easier than browsing a tree of accounts (you can filter all accounts of class 30*, filter on the name to access quickly, group by Type, …)

The default way to browse accounts SHOULD NOT BE anymore through the Charts of Accounts tree view. The user will browse the data through the HTML reports (P&L, BS or Charts of Accounts) that have a hierarchy.

Have a look at the reporting section below to understand this.

Legal reports WILL have a hierarchy but defined by the report (Profit, Loss...) We keep the list view of accounts for the Configuration section. (not for a daily usage)

We keep the aggregation system for consolidation of multiple companies (we will also add a percentage on the consolidation report for companies that aren’t owned at 100%). We add “Tags” on accounts → the tags will be used for reporting and, eventually, the hierarchy in the CoA.

TODO:

- Remove “parent accounts” and “children accounts” on account.account object
- Change the way credit and debit are computed on accounts (no need to sum children accounts)
- Remove the type: “View” on account forms
- Root account of chart template disappears. Every account template has chart template like taxes.

- Add a new field many2many('account.account.tag') to a new object having the following fields:
    - Name
- Define an on_change method to automatically select the type and the tags based on the code: assign the tags and type of the account having the biggest number of digits in common and minimum 2 (if less than 2, don't assign anything to these fields.)
- Remove the "Chart of Accounts" menu. (the tree view on accounts); the list of accounts is enough. Moreover, we will have a General Ledger report that renders a chart of accounts with a hierarchy. (new reports are on screen with filters and options like comparisons.)

To understand why it's easier to work on lists of accounts, you need to understand the task that improves search, filter and groups. You can test the related mockup here: http://k2lkjf.axshare.com/#p=advanced_search&hi=1&c=1 (this is a dynamic mock up, try to click on the expand filter icon)

It's super easy to filter or group by account types (and this creates hierarchies too).

Active accounts (DONE)

The concept of "Active" is too complex, and refers to something uncommon for accountants. We propose to replace that concept with the following rules and measures:

- The user can delete an account if it has no journal items
- If there are journal items, we can not delete it
- If there were journal items in previous fiscal years but not in the current one, this account does not appear in reports, unless the account is involved in the dates covered by the report (YoY comparisons)
- Replace "Active" by "Deprecated" checkbox; the account is still visible but not accessible in many2one linking to accounts.

That way, the same features are available but the usability is improved.

TODO:

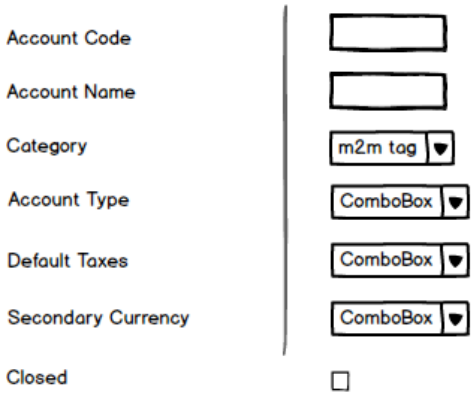
- Remove the active field on accounts (you can delete an account, but not unactivate it.)
- Create a "Deprecated" checkbox
- Remove the "Close" account type
- Put domains on all many2one linking to accounts to not see "deprecated accounts".

Account Types (DONE)

Currently, an account has two types:

- Account Type: used technically (views, aggregation, receivable, payable)
- User Type: depends on the country, used for legal reports (P&L, BS)

In v9, an account definition form should only contain one field: the User Type (renamed into ‘Type’). The current Account Type should be a field on the User Type object.

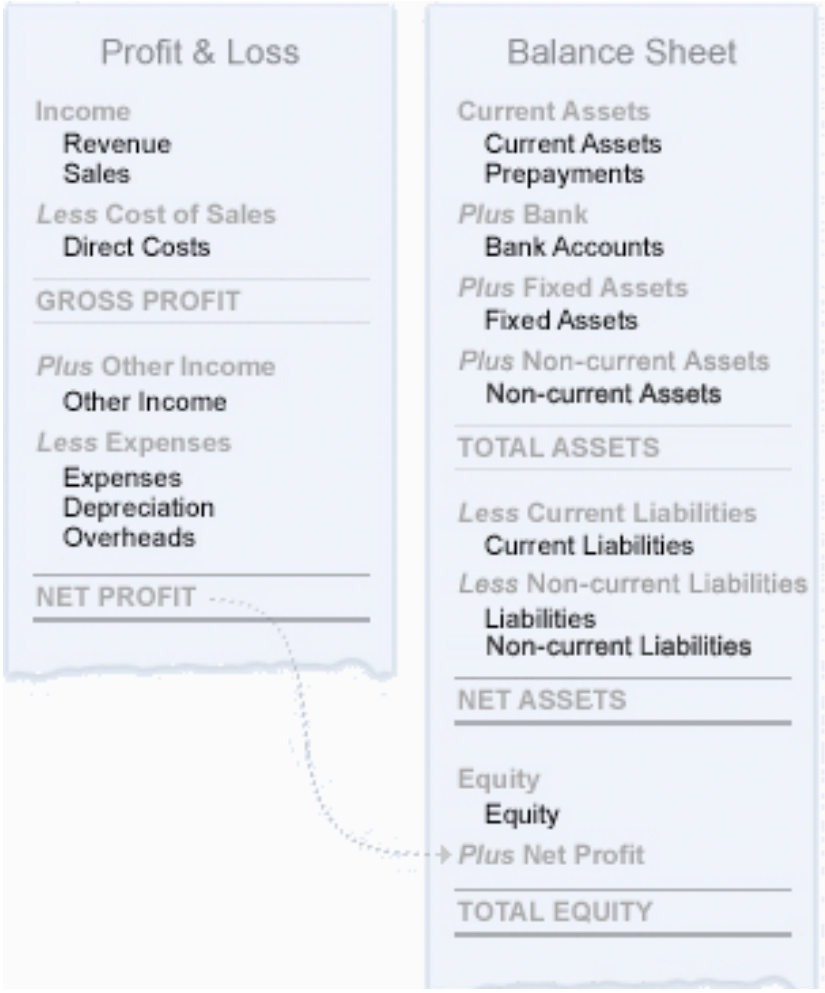


Note that people will still be able to continue defining their own “user type” through the configuration or localization module if they want but we believe this won’t be necessary most of the time as we plan to create by default the main ones (revenue, sales, direct costs, other income, expenses, depreciation, overheads, current assets, prepayments, bank accounts, fixed assets, non-current assets, current liabilities, liabilities, non-current liabilities, equity).

TODO:

- On account.account.type:
    - Put a “type” selection field, with same definition than the current one on account.account
    - On account.account, the type field becomes a related to “user_type_id.type”

- Remove the Type field from the form view
- create in the account module, the following account types: revenue, sales, direct costs, other income, expenses, depreciation, overheads, current assets, prepayments, bank accounts, fixed assets, non-current assets, current liabilities, liabilities, non-current-liabilities, equity)
- configure the built-in profit / loss and balance sheet reports as shown below:



Journals (DONE)

Journals should have two sequences:

- Invoices (already exists)
- Refunds (new many2one field to add)

This allows companies to put invoices and refunds in the same journal or create two separate journals.

Advantages of putting in the same journal:

- Turnover is correct in the list of invoices (invoices - refund)
- Dashboard is simplified (not 2x every journal)

By default, at the installation of the software, only one sale and one purchase journals are created (invoice and refund are in the same journal). This allows to have a clear dashboard and not duplicate every menu.

When creating a refund based on an invoice, if the sequence is set for a refund, the system do not ask for a journal and automatically put the refund in the same journal (in the wizard).

In the dashboard, the button to create an invoice has a dropdown to also create a refund. Like this: http://getbootstrap.com/components/#btn-dropdowns-split

Invoices (DONE)

Add a computed field on invoices: amount_total_signed which is the amount in the currency of the company and negative for refund invoices (or refund supplier invoices). Also review the invoice analysis report to simplify it.

Display this field in list views (instead of the current amount_total one) but not in forms views. This would allow to put invoices and refund invoices in the same menu and have the total below which is correct.

The onchange to set total and taxes should be automatic. This would be an automatic win when porting to the new API.

Change data/demo data → the default is to use the same journal for invoices and refund rather than two different journals. People could change this but by default we will put invoices and their related refunds in the same journal.

Payment Orders (DONE)

Payment should be managed by the “Pay Invoice” button on supplier invoices, instead of the current Payment Order form. Once invoices are paid via this button, a menu can be used to send the payments to the banks. (or generate files like SEPA).

So, payment order are replaced by journal entries with a status. You must be able to search / filter on invoices based on their payment status: To Pay, Payment Ordered, Payment Done.

- Remove the account_payment module
- Put a “payment_state” field on journal entries “account.move”
    - None
    - To do
    - Done
- On the journal, if it’s a bank journal, add a:
    - payment_type field selection:
        - Manual
        - Check Printing
        - SEPA (if account_sepa is installed)
- In the Pay invoice wizard, display a selection field, with default coming from the journal:
    - Pay via: SEPA, manually
- In the payment order menu, display all journal entries having payment_state in “to do”:
    - Domain: payment_state <> None
    - Default Search: payment_state = “to do”

On supplier invoices, add a wizard allowing to mark selected invoices for payment in batch. Need to have ability to select supplier credits as well, so when the invoices and credits are selected, the wizard need to show for any invoices with outstanding balances after the credit is applied
This allows to do “Bulk Payments”. This wizard lets you select:

- Payment mode: Journal
- Payment Method: SEPA, Check, … (come from the selected journal)
    - near the payment methods, you have information like:
        - Current Balance: 4500€
        - To Pay: 2300€
        - Scheduled for Payment: 1000€
        - Balance: 1200€
- Date: Directly, At Maturity Date
- Need to have an option to send remittance advice to suppliers (to be integrated in the print check module)

Remove model of journal Items (DONE)

We plan to keep model of journal entries, but not subscriptions. The subscription mechanism will be replaced by:

- contracts: for invoice based subscriptions (already exists)
- model of journal entries: that can be called easily in a journal
- model of journal entries in bank reconciliation (already exists)

Remove (temporarily?) subscriptions on journal entries. The feature is not good enough to be maintained as it is. More over, the new tools allows to better manage recurring entries:

- bank loans: use models during bank reconciliation process
- rentals: you can use recurring invoices on contracts
- payroll: use model of entries and change them once a month
- social taxes: same than payroll.

As it will be very easy to call a model or use models during bank statements reconciliation, I think we don't need subscription on journal items anymore.

TODO:

- remove the objects
- add a boolean field on journal entries: "Journal Entry Model"
- Add a button on journal entries form view to call a model, select a model in the same journal and the journal entry s automatically created.
- Add a "save as a model" wizard on journal entries (you can get it from the More… menu)

note: we removed both the model and the subscription (temporarily)

Notes for New API conversion (DONE)

to review again based on the decisions made in the other sections
Some technical comments to clean when porting to the new API.

account.py (DONE)

- remove method check_cycle (since we remove parent_id)
- payment.term:
  - The latest row of a payment.term should always be "balance"
- payment_term.line
  - Values should be in %, instead of between 0 and 1
- account.account.type
  - remove the following method (and check why we needed it)
    - _get_financial_report_ref
    - _save_report_type
    - _get_current_report_type
  - close_method:
    - remove the "Detail" one
    - check if we still need the "unreconciled" one
- remove method _code_get (use many2one instead of selection field for type)
- account.account object:
  - remove parent_id, parent_left, parent_store
  - remove _get_level, _get_child_ids, _check_recursion, _check_type, _check_account_type, _check_company_account

    - simplify name_search
        - remove the feature that allow to search on “name:...” or “type:...”?
    - name_get
        - according to the context, name_get should return
            - Code (only)
            - Code Name
    - remove copy method (and possibly set copy=False on some fields)
    - remove _check_allow_type_change
    - remove _check_allow_code_change (forget about the message about lack of confidence. People will like Odoo because it’s simple.)
- account.fiscalyear:
    - remove code field, keep only name
    - remove name_search method
    - rename end_journal_period_id into end_journal_id (many2one to journal,)
- remove object account.journal.period (and everything related to it)
- account.move object:
    - remove account_move_prepare (this method is not used anywhere)
- account.move.reconcile:
    - remove line_partial_ids field
    - remove type field
    - remove constraint _check_same_partner??? to discuss i in workshop
    - remove reconcile_partial_check
    - remove name_get (since there is no partial anymore, no need for a specific name_get)
- ~~remove account.model and account.model.line (and everywhere it is used)~~
- remove account.subscription and account.subscription.line (and everything related to this)
- remove account.

account_analytic_line.py (DONE)

- remove the search() method. Check if some methods are using (from_date or to_date) in the context and use a default_search on the context instead.

account_bank.py (DONE)

- check why we need post_write? This code should probably be moved elsewhere; in the code the create the bank and journal

account_bank_statement.py (DONE)

- Separate account.bank.statement and account.cash.statement into two objects (eventually inheriting from the same object)
    - A lot of mess is coming from this

- account.statement.operation.template
    - We may need both a fixed amount and a percentage amount (AND or OR)

account_move_line.py (DONE)

- remove method default_get_move_form_hook
- remove convert to period
- default_get should be simplified A LOT;
    - no need to compute remaining based on debit/credit/taxes
    - remove default_get, put default methods on fields
- remove field invoice
- remove list_periods, list_journals
- (probably more simplification to do here)

module structure (DONE)

Remove the subfolder account/project and put everything related to analytic accounting into a single .py and .xml files.

Change everywhere: (DONE)

- the context should never contain period_id or journal_id!
    - use default_period_id and default_context_id instead

Wizards (DONE)

Remove the following wizards (and everything related to them):

- account_automatic_reconcile.py
- account_change_currency.py
- account_chart.py
- account_journal_select.py
- account_reconcile_partner_process.py
- account_subscription_generate.py
- account_use_model.py

Misc (DONE)

On account.move, add a fields.reference('Origin') that is the real document that created this entry:

- sale invoice, purchase invoice, sales receipt, purchase Receipt, Asset, bank statement
- If nothing in reference, it's a Journal Entry

Remove the decimal precision "Account" and instead of it,

- round by using the currency (foreign currency involved or company currency)

- store the full length of the decimal in db
- use widget=monetary in views so that the display of these fields are still fine

Bank and Cash Statements (DONE)

The current implementation of bank statements and cash statements is ugly. I would suggest to split it in two different objects so that they can each implement it's own logic without having side effects.

Refactor the code so that:

- account.statement & account.statement.line becomes the main object:
  - rename object account.bank.statement → account.statement
  - rename object account.bank.statement.line → account.statement.line
- account.cash.statement _inherit**s** from account.statement and add his own logic. Change the menus and screens to directly work on the account.cash.statement for cash registers.

Expenses (DONE, yes? Ask BST?)

In the v9, employees will directly create expense lines (or expense lines can be created by users on behalf of others employees) instead of creating expense sheets.

Here is a proposition of a new user flow:

1. Users create expenses
2. They can click "Submit" on expenses:
   a. when submitted, the expense "line" is added to the expense sheet that is still "to validate" for the same user.
   b. If no expense sheet "to validate" exists, a new one is created
3. The HR/Manager can validate expense sheets that are in "to validate"
4. The accountant can generate entries for validated expense sheets

TODO:

- rename module hr_expense → expense
- rename object: hr.expense.line → expense
- rename object: hr.expense.expense → expense.sheet
- add a state field on expense object: Draft → Submitted
- add a button "Documents" on expense sheet to open the attachments of related expenses "e.g. scanned documents"
- Create a mobile app to easily:
  - Take a photo of an expense

- Register name & Price
- Send to Odoo
- Track status of expenses

The tax(es) should also be selectable on the expense line.

Reporting Engine (DONE)

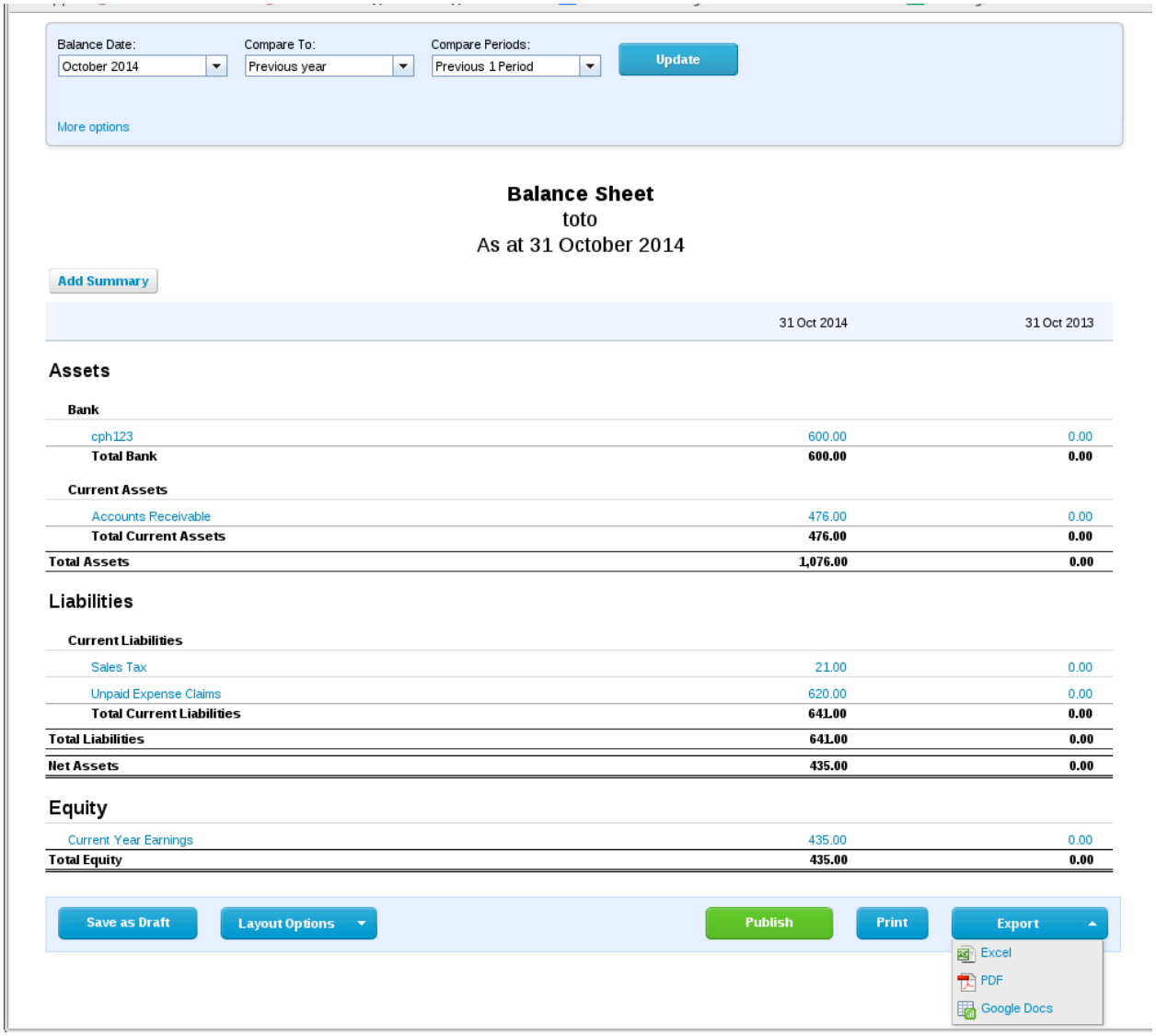
In the past, we had two different usages supported differently in the application:

- List/Tree views to browse data in the screen (e.g. tree of accounts)
- PDF reports to print data (e.g. P&L and BS)

In the future, reporting should become the default way to browse the data. Reporting will be normal HTML views that opens in the normal UI (keeping the left menu). Wizards are replaced by filters and options on the top. Whether you want to work on the data (in the screen) or print them; it's the same.

For performance issues with huge reports (e.g. general ledger), we can use pager and lazy loading to make them very fast. Of course, if you click on "Print" button, it still uses the webkit print (with everything unfolded)

On the technical point of view, reports will be generated server side. That means that if we want them to be dynamic, we need to split it into severals small templates and when doing an action (like fold/unfold) we can only call these small templates and replace them inside the generated html (this in order to prevent reloading the all report whenever we change something)



TODO: improve the mockup. It's not good enough. The reporting has to be a killing feature. We have to do 3 sample reports in mockups to go into the details of the features. E.G.: we can use graphs on top of the data table (like in google analytics) put postits everywhere on the page, add sparklines near account numbers, ... We should brainstorm a bit more. what about using pivot tables? No accounting software has a real BI integrated.

We will add actions in the reporting. (e.g. you can reconcile journal items directly from the partner balance report) If you find journal items to reconcile or if you want to drill-down in an account, you can do it directly from the report.

The filters are on the top. You'll be able to change the options and will see the result in the same page, or expand some part of the report as some lines will be clickable (e.g. click on an account to zoom into it)… We can implement fold/unfold in the report as well.

The idea is to offer the possibility to users to play with the option until they get the information they need. Then, they have a button to print the current report displayed in a pdf.

We can also add features like “Add a Comment” on every line/header/footer. (like post-it) This way, the accountant can comment a report before giving it to the managers. Simple example: Print the Aged Receivable report and put notes for the responsible of the credit collection effort. (check why this customer did not pay, he told me it should have been done last week)

List of features on reports:

- filters / group by on the top of the sheet
- Add notes everywhere
- Fold / Unfold (example P&L, BS) in JS
- Drill-down (click on an account to get it’s detailed journal items)
- optional graphs and/or sparklines
- We need extra fields like tags on the account that can be used to group.
- We should be able to create groups that are master data independent of the chart of account in which we put the accounts we want to analyse together. Especially if there is not much done in analytic accounting. These groups should be used as a selection criteria and also a grouping criteria in reporting.

Development in progress on
https://github.com/odoo-dev/odoo/tree/master-html-report-configurator-pvy

Even if some options of the filters are common (YoY comparison, accrual/cash based), some options should depend on the report, example:

- P&L Dates: from … to …
- BS date: At …

Reports to add:

- Executive summary (ratio like the “Besoin en Fond de Roulement (Working Capital), disponibilité de trésorerie, ACID test…)
- Chart of Accounts / General Ledger (note: it looks like the term General Ledger is not the same in U.S. or in Europe: In sage, the general ledger is a print of all journal items per account. In Xero, it’s all accounts and their balance, but you can click to zoom to the journal items)
- Cash flow statement
- Tax Summary (kpi to control the tax report is correct)
- Partner statement (an improved Aged Balance)

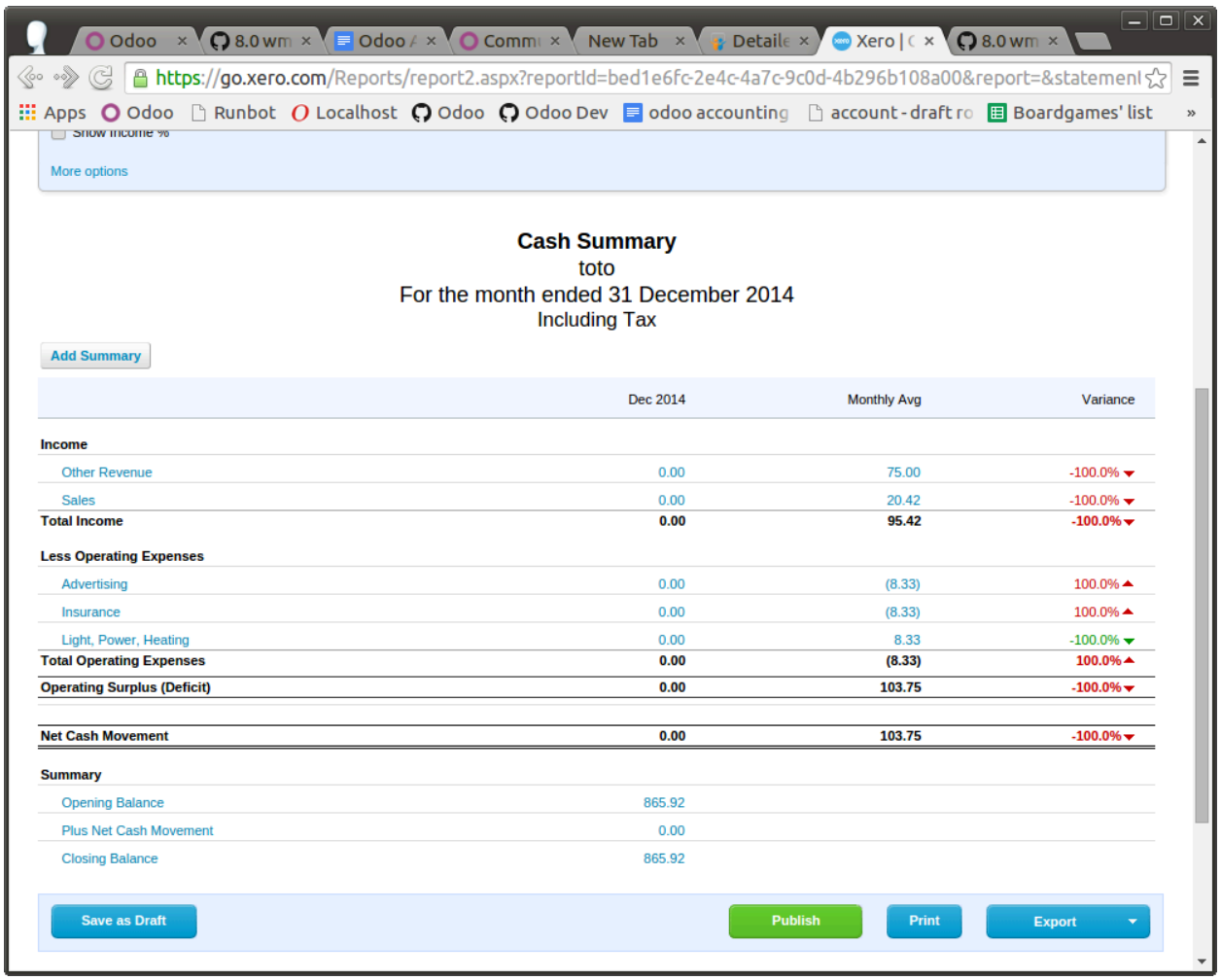
Replaced :

- the General ledger will replaced by the list view of account_move_line in order to be able to have more flexibility (search, group_by, etc)

All other reports have to be improved: aged balances, p&l, bs,

Implement the Accrual/Cash Based as an option to most reports.

cash flow statement example:



Reports Metadata (DONE)

currently being written

Some accounting reports should be defined using account financial report: these are the P&L, BS, Chart of Account (Trial Balance), Cash Flow Statement, Executive Summary and all the country-specific reports. All the other reports (tax generic, aged partner balance, followup,...) will use custom code and won't be based on account financial report.

Account.financial.report should be based on filters, that way we can specify it for any account or other financial reports.This could be used to create hierarchies in the Chart of Account report

(not anymore a tree view) or to create KPIs in the executive summary like the “Besoin en Fond de Roulement” / “Working Capital”.

We will split the current account.financial.report in 2 models: account.financial.report and account.financial.report.line. The first one is there to defined the report and global options, such as number of columns inside the report (debit+credit+balance or only balance), header, and so on. It will serve the same purpose as the current wizard we have when opening a report. The second model correspond to the current account.financial.report

Should be generic for all reports:

- Target moves (all entries, posted entries)
- date_from
- date_to

Object: account.financial.report

- Boolean: allow comparison
- Boolean: allow cash basis method
- Boolean: show debit-credit
- Boolean: show balance
- Report Title

Objet: account.financial.report.line

- Domain (to specify which journal or account to show) (should use ir.filters)
- group by
- Reference field: Action to perform when clicking on a line (open account, invoice, other)
- Type: selection:
    - ~~Accounts → to be removed, it’s more flexible to force using tags~~
    - Account Types
    - Account Tags
    - ~~Reports → to be removed, we can use formulae for this~~
    - Sum of Children
    - Formulae

- Formulae
    - e.g: [Income] / [Income,-1] * 100 (income growth)
    - Can be based on
        - [ReportCode]
            - [ReportCode,-1,debit] → preceding period analysed
            - [ReportCode,-1:-3,debit] → preceding 3 periods analysed
            - [ReportCode,0,debit] → current period
        - {tag} → not sure if ReportCode is enough or not?
            - {tag,-1} → same than ReportCode
        - #Days → Number of days in the analysed period

            - #Days[-1,-3] → Number of days in the last 3 periods
        - #Months → Number of months in the analysed period
- Report Type:
    - Currency, Percent, Simple Float
- Display Detail
    - Sum Only
    - All Accounts
    - All Journal Items
- Drill Down:
    - Show ….
- Style
    - h1,h2,h4,h4,p
- Possibility to have python code (with a safe eval), that way we can do things like if total > 0 then print something else show nothing
- Should be able to print tax declaration too

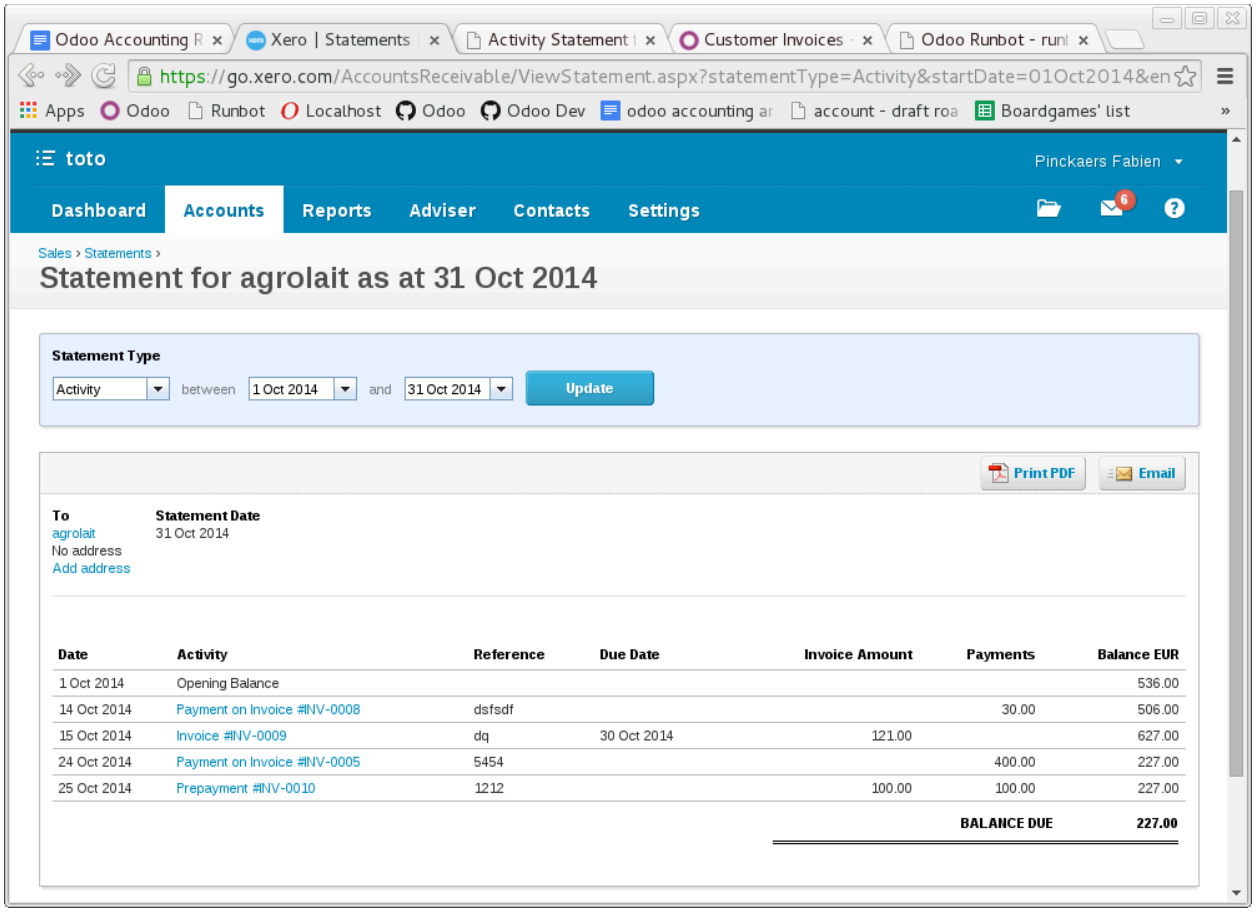
TODO full spec
On every report, when clicking on a line, it should open the right document, according to the origin reference field on the account.move.

Followups Review

The followups are going to be improved in the following way: account main module will have a simple followup system made by invoice that should be enough for most of people. The advanced features will be added by the current account_followup module.

“account” module: (DONE)

On the list view of invoice, allow to easily add an expected payment date. Order the list of unpaid invoice by expected payment date (first the ones without expected date), highlight in red the overpassed ones. On the same invoice list, you can easily log an internal note or do the usual stuff (print the selected invoices, send them by mail..). The new thing is that you have the possibility to sent the “customer statement”. That action make you jump on the list of partners have a receivable account > 0 and showing their due amount and last reminder sent date. You can select only some of them and effectively print the customer statement which list their invoices and payments under a static customizable text (to set on the company). The report is HTML made and first displayed on the screen before you’re actually able to print it.



“account_followup” module : (DONE)

Once this module is installed, users can use the full power of the current module: define several followup level with their level-specific text to print on the customer statement in place of the regular text and accordingly to the highest level of the partner’s invoices.

New features

kanban journal as a dashboard (DONE)

Menus are too complex. With a clean kanban view, the user will not even need to use menus. A lot of menus may be removed due to this. (function like import Bank Statement can be done directly from the Kanban view with a big button on the Bank journal).

The kanban view of journals will help to reduce the number of menu items by providing some info in a handy way and shortcuts dedicated to the type of journal. When you click on the “Accounting” menu item, you arrive in the dashboard (=kanban view of journal)

Each journal will show various action and information as well as shortcuts for the user. Here is the list for each journal of the actions it needs to be able to perform:

Sale journal:

- a 2-lined chart showing the current month turnover against the turnover of last month
- figures and shortcuts will be for the sale journal AND for the associated sale_refund journal
- How many (draft, due this month, to be send) invoices we have
- Link to see/create invoices
- payment followup actions

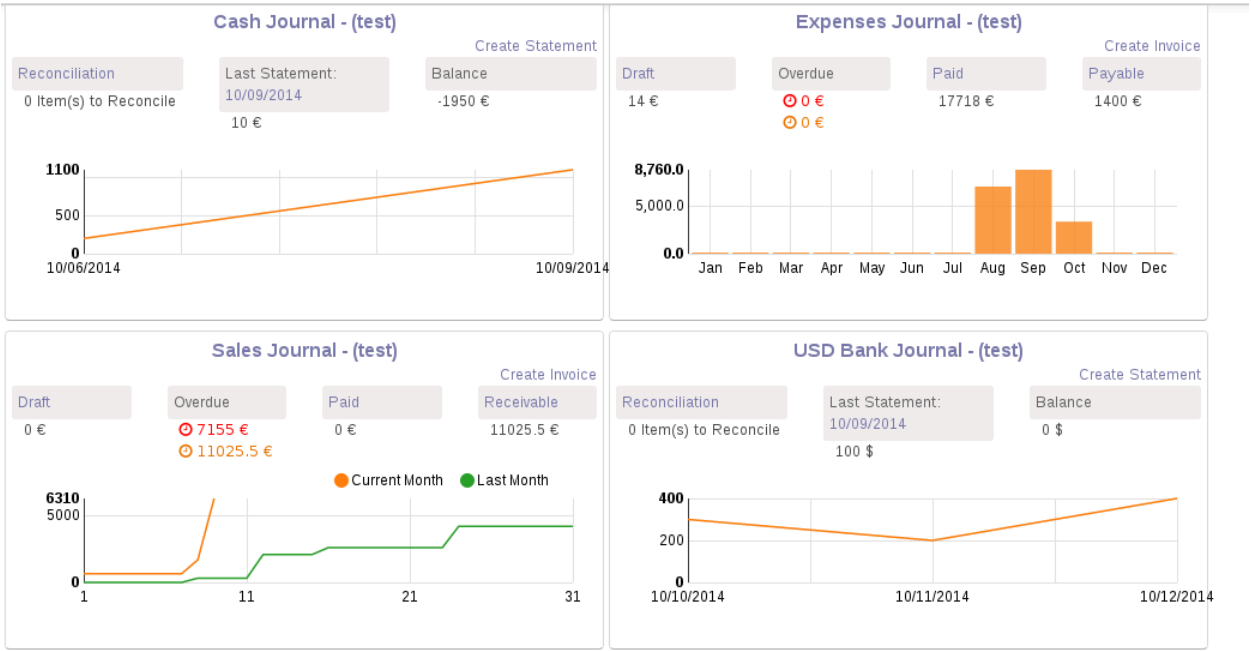
Purchase journal:

- a graph showing the estimated cost over time. graph should start today and value should go increasingly for the future (as we have draft and validated purchase that will come)
- figures and shortcuts will be for purchase journal and associated purchase_refund journal
- show how many draft, due, validated purchases we have
- Link to see/create purchase invoice

Bank and cash journal:

- Line graph showing journal entries
- Transfer Money → New wizard to develop, to create a journal entry
- Send Money → Purchase Receipt with default to the bank account
- Receive Money → Sales Receipt with default to the bank account
- Reconcile Account
- Import a Statement
- View/create statements

can be tested on : https://github.com/odoo-dev/odoo/tree/master-journal-dashboard-csn

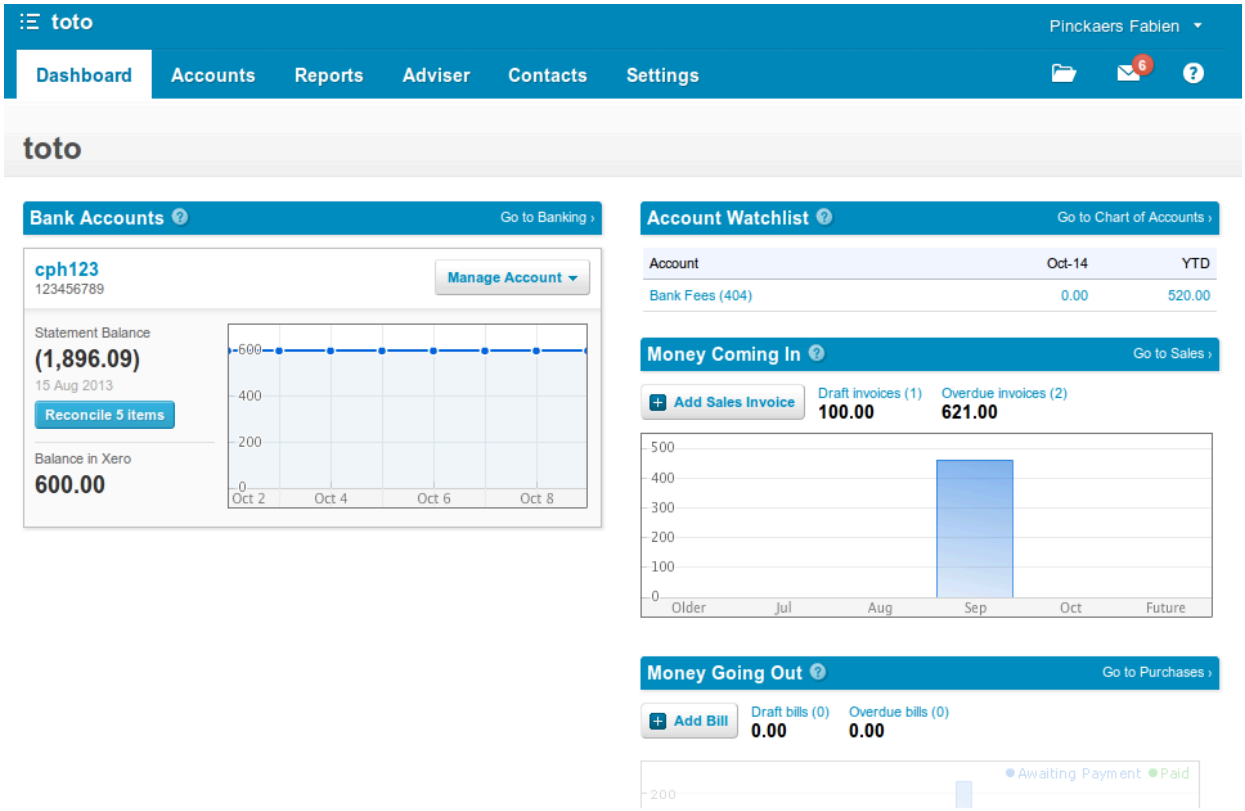


The designers will make this dashboard beautiful.

Note that:

- the graphs should only be visible for the managers of the accounting application (to deal with access rights restrictions) → add the group in the kanban view
- in the kanban view, users will only see journals for which they have been added as users (as they would do for projects)
- only “Advisor” group can see the graph

Here is how it looks like in Xero: (main buttons like Reconcile on bank statement journal or “Create Invoice” on sales journal are directly in this view)



Note about money transfer:
A traditional approach to transfer money from one bank account to another is to use an intermediary account (e.g. in Belgium, 58 virements internes) But most accounting software do not have a great bank reconciliation feature: you can pre-register a transaction and it's reconciled afterwards when receiving the bank statement. But with a good bank reconciliation mechanism, it changes everything.

I just tested Xero and they do a direct transfer that keeps a "unreconciled" status. It's at the same time more powerful and easier:

- More powerful: as you directly give the right destination account, the system expect to get this transfer in this specific account. (forecasts are better, reconciliation is easier)
- Easier: no need to worry about intermediary accounts and reconcile entries in this account. It's faster, you only have one transaction to record, the reconciliation is done a the statement of the second bank. Moreover, intermediary accounts are a nightmare in multi-currencies. (how do you know if the difference is due to currency exchange rates or profit&loss when reconciling afterwards)

So, the money transfer wizards should only have the following fields:

- Date
- From Account
- To Account

- Amount
- Communication

It should just create a Journal Entry. (directly from one bank/cash account to another)

It can be against the habit of some accountants that are used to use an intermediary account, but they can still do that during the bank reconciliation process (even in one click using model of entries)

Banking flow (DONE)

As said in the introduction, one of our primary objective will be to smooth the processes related to the banking flow. The import of CODA, .OFX and .QIF files have already been developed. The SEPA integration is also in the pipe. (direct debit, payments, bank statement import).

But we want to go much further than that. We want to have bank statements to be imported automatically from the bank every hour. We want payment order to be sent automatically to your bank, checks to be printed automatically, …

Fabien started negotiation with external web-services in some countries to automate these interfaces with banks.

What does every manager do when he arrives at the office? He wants to check the payment received on the day/yesterday. We want Odoo to allow that! A real time visibility on your cash and the payment of your suppliers in one click, …

This will be the reason why people will love Odoo accounting. It will even be in our new user on boarding flow (see below)

Reverse entries wizard (DONE)

Years ago, a module named account_reversal was created by the community. Its goal is to offer an easy way to create the exact opposite accounting entry than the selected one.
https://www.odoo.com/apps/7.0/account_reversal/

We are willing to integrate the main feature of the module as part of the core features of the accounting, but we believe it needs

- some code review
- code conversion to new API
- some of the satellite features (e.g. checkbox “to be reversed” + filter) to be dropped

Storno and Anglo-Saxon

Storno (TODO)

The Storno module (putting negative values in debit and/or credit when reversing a journal entry) should be integrated by default.

Anglo-Saxon review (DONE, in PR phase csn)

Both Storno and Anglo-Saxon should be in separated module AND they must be activated by company. This would allow to have multi-companies environment, some with storno accounting, others without.

Modularity (DONE)

The following changes should be done on modules:

- merge account & account_voucher (and remove all the unnecessary features in voucher)
- put Python Code for tax computation and “Rates” in a separate module: account_tax_advanced

Tests & Demo Data (DONE)

Tests and demo data currently rely on a dedicated test chart of account and taxes, which currently results in creation of duplicated Chart of Accounts, duplicated journals and so on. Those demo data pollute the demo instances and confuse people. The runbot is also hardly usable because of several localization modules installed.

In order to clean the databases installed with demo data, and the runbot, we gonna do the following changes: the demo data and the test files will be located in the “account” module but they will only be instantiated and tested in localization module: they will simply be referred in “l10n_*/__openerp__.py” by

‘demo’: [“../account/demo_file.xml”]
‘test’: [“../account/test/test_file.xml”]

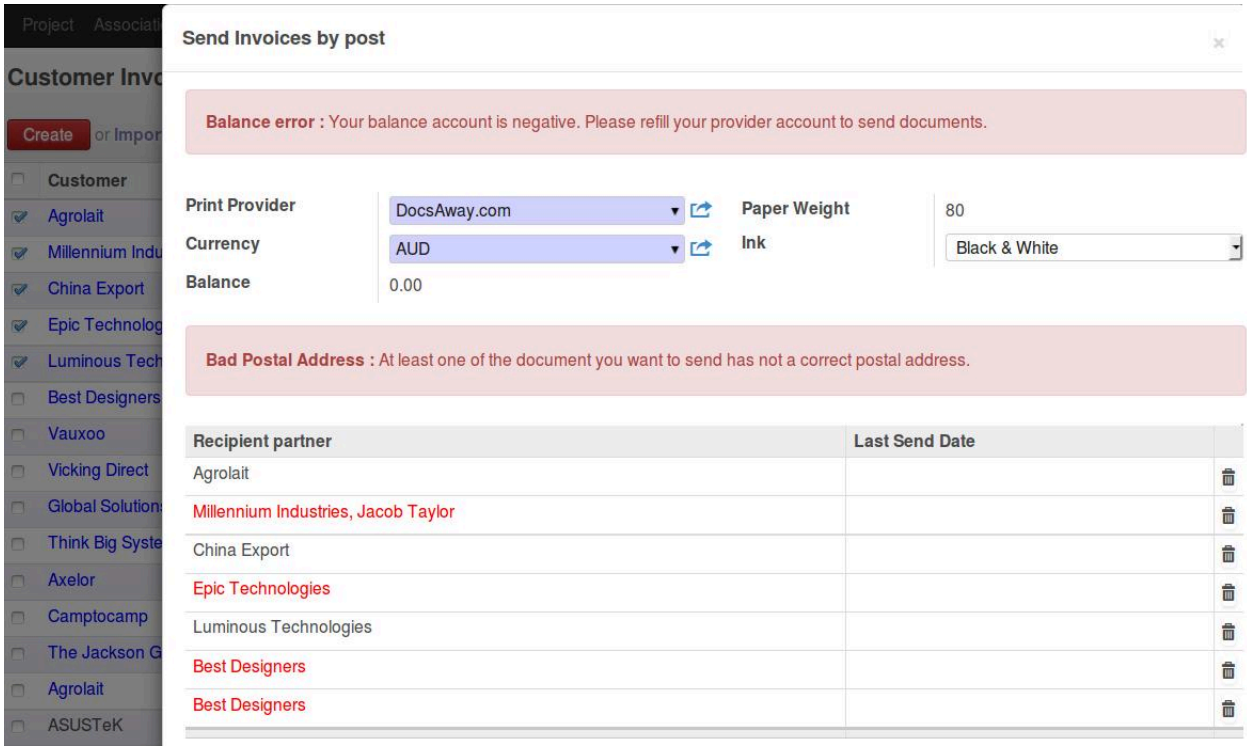
When installing a l10n_* module, the system will check if there already exists a CoA for the user’s company. If not, it will directly generate the CoA from the installed templates, otherwise the template are just installed normally.

In order to have the accounting tested using a random localization, the runbot will install only one random localization module (for the main company), skipping the other ones.

Journal Entries Recording (DONE)

Send Invoices by Regular Mail (DONE, to be merged)

We developed an interface with a service that allows to send invoices/quotations by regular mail in all countries. (invoices are printed in the local country and delivered by the post. It costs less than $1 per letter to send). We use the services of Docsaway and have an agreement with them to put the Odoo logo on the front letter.



TODO: improve this mockup: Settings like Ink, Paper Weight, currency, balance can be in the Settings. (unless the balance is not enough then we should have a message)

So, you don’t need to print invoices, put them in envelop and post them. Just send them by post in one click. The wizard works on a single invoice but also allow to send a batch of invoices at once.

Add Management Reports (DONE)

Executive Summary
toto
For the month of October 2014

Add Summary

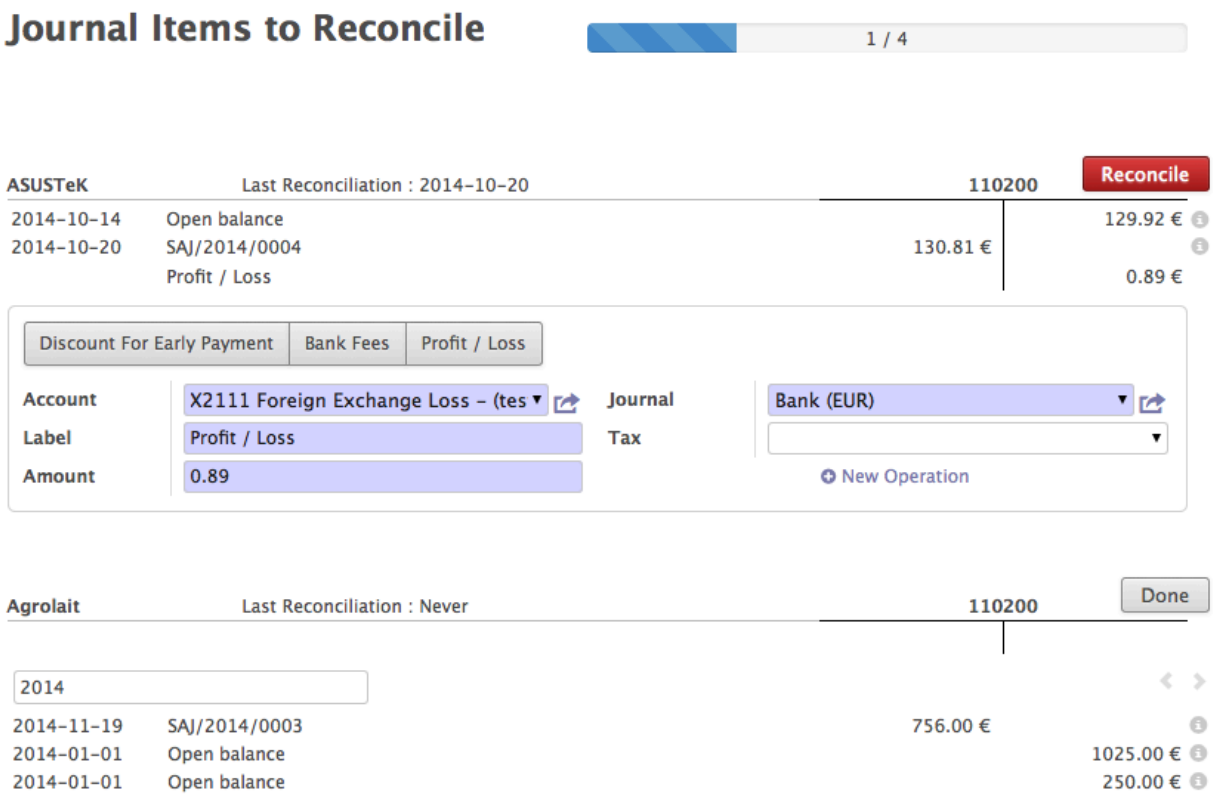
	Oct 2014	Sep 2014	Variance
Cash			
Cash received	0.00	640.00	-100.0%
Cash spent	0.00	40.00	-100.0%
Cash surplus (deficit)	0.00	600.00	-100.0%
Closing bank balance	600.00	600.00	0.0%
Profitability			
Income	0.00	1,055.00	-100.0%
Direct costs	0.00	0.00	0.0%
Gross profit (loss)	0.00	1,055.00	-100.0%
Other Income	0.00	0.00	0.0%
Expenses	0.00	620.00	-100.0%
Profit (loss)	0.00	435.00	-100.0%
Balance Sheet			
Debtors	476.00	476.00	0.0%
Creditors	0.00	0.00	0.0%
Net assets	435.00	435.00	0.0%
Income			
Number of invoices issued	0.0	4.0	-100.0%

Improvements of existing features

Access rights (DONE)

Rename access rights: Billing, Accountant, Adviser.

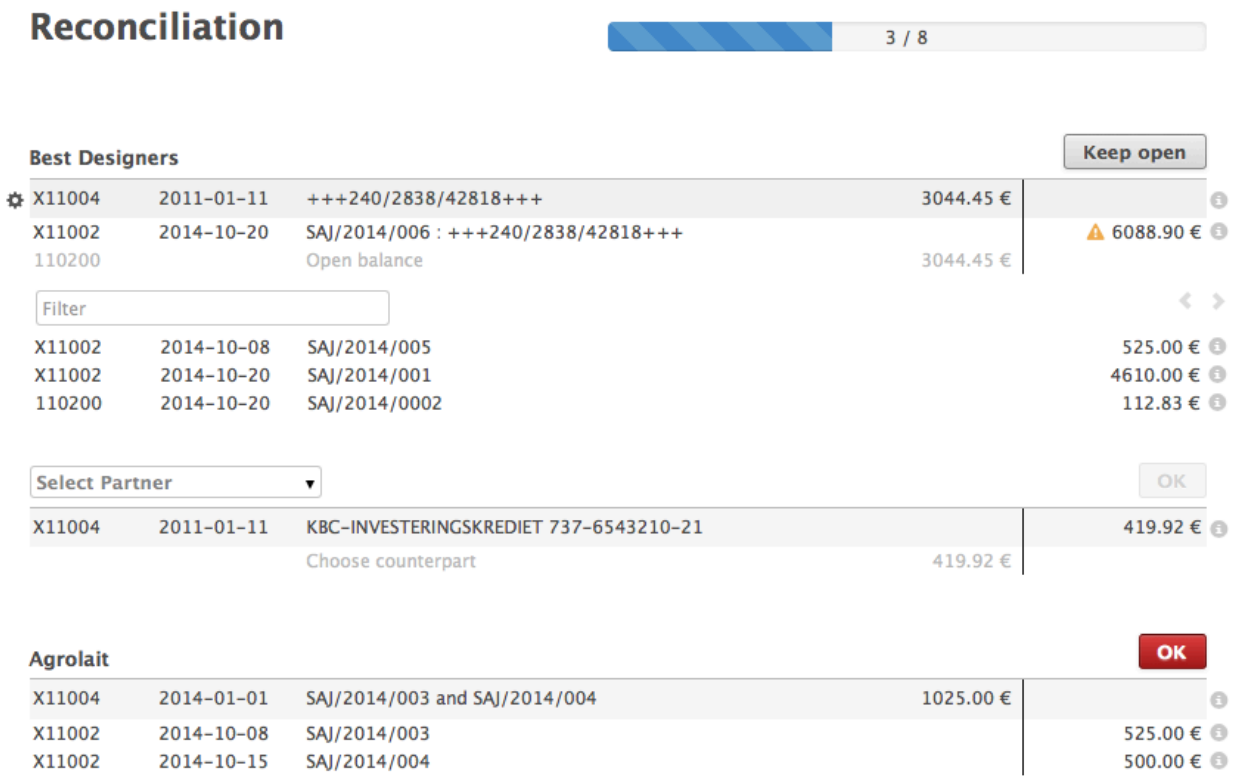
Reconciliation (DONE)



WIP : https://github.com/odoo-dev/odoo/tree/master-manual-reconciliation-ama/
in the window of manual counterpart, add a gear to save as a template and another to manage templates. If the list of button is too big, have a button more instead of going in several lines

Since bank statement do not use the account.voucher object anymore, it can be simplified a lot. Account vouchers are used for Sales Receipt and purchase Receipt only.

Bank Statement Reconciliation (DONE)

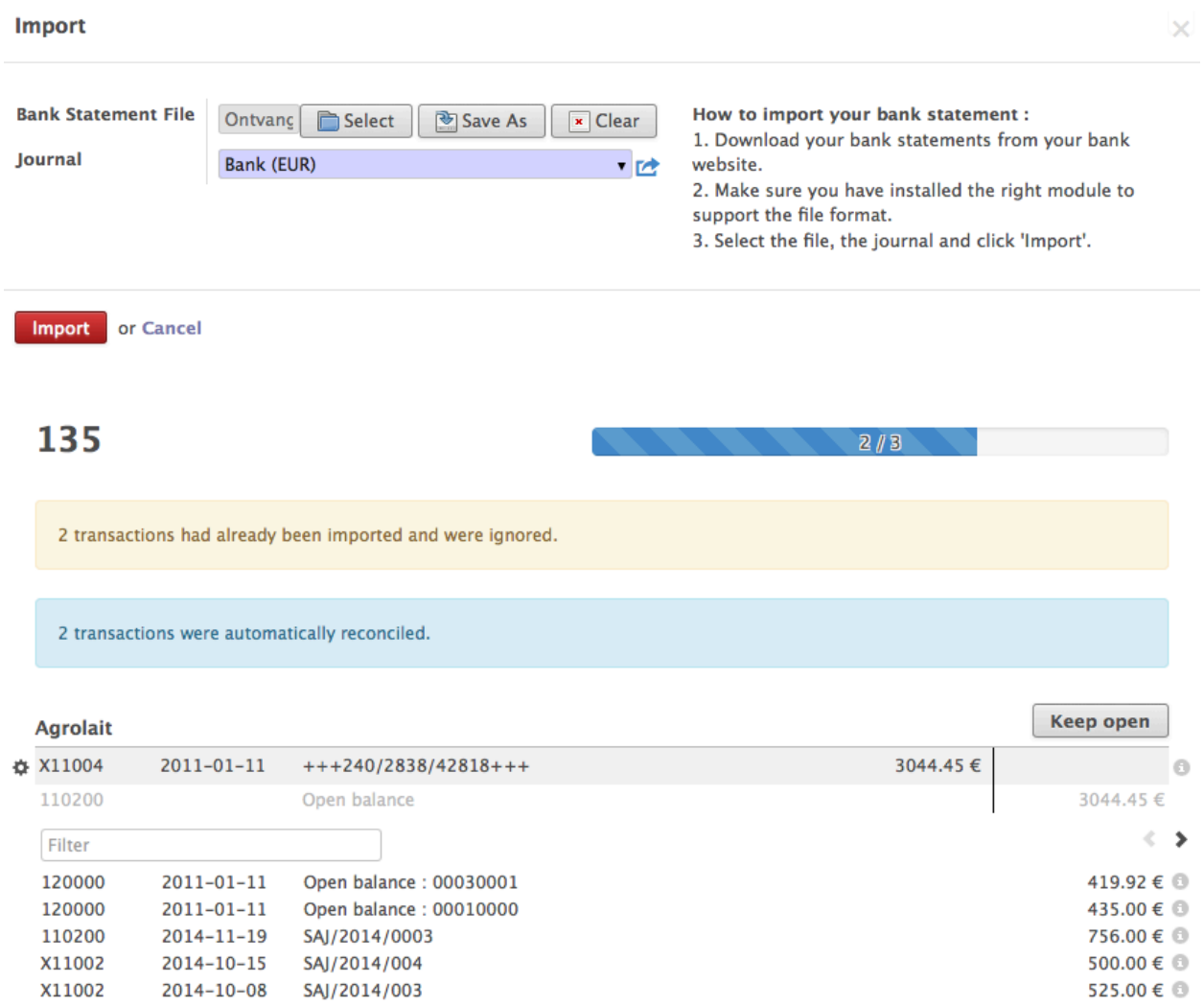


To check:

- Don't remember if we finished the "Reconcile With Existing Entries" feature (if the guy used the Pay Invoice button. Useful for checks handling (or credit card payments). → Some countries prefer to do that rather than using an intermediate accounts.

Bank Statement Import (DONE)

Import of OXF, QIF, CODA (DONE)



WIP : https://github.com/odoo-dev/odoo/tree/master-manual-reconciliation-ama/

Todo:

- Improve the binary field widget (ugly and unusable)
- Reconcile the firsts entries and the rest in the backgrounds to not lock the UI
- in the notification window:
    - add links to go on the list of items
    - do not close by clicking on the notification but only if we click on a cross

For information, the standard flow of reconcile statements in the U.S.: Check the user flow.
Todo: check that the dashboard (journal kanban) display these information.

Currently supported format: CODA, QIF, OFX.

Import CSV files (TODO)

Make it work for the CSV import too → based on the CSV import tool. Just create an act_window that opens the bank statement lines (in dashboard's menu) so that you can press on "Import" link.

Menu items (In progress csn)

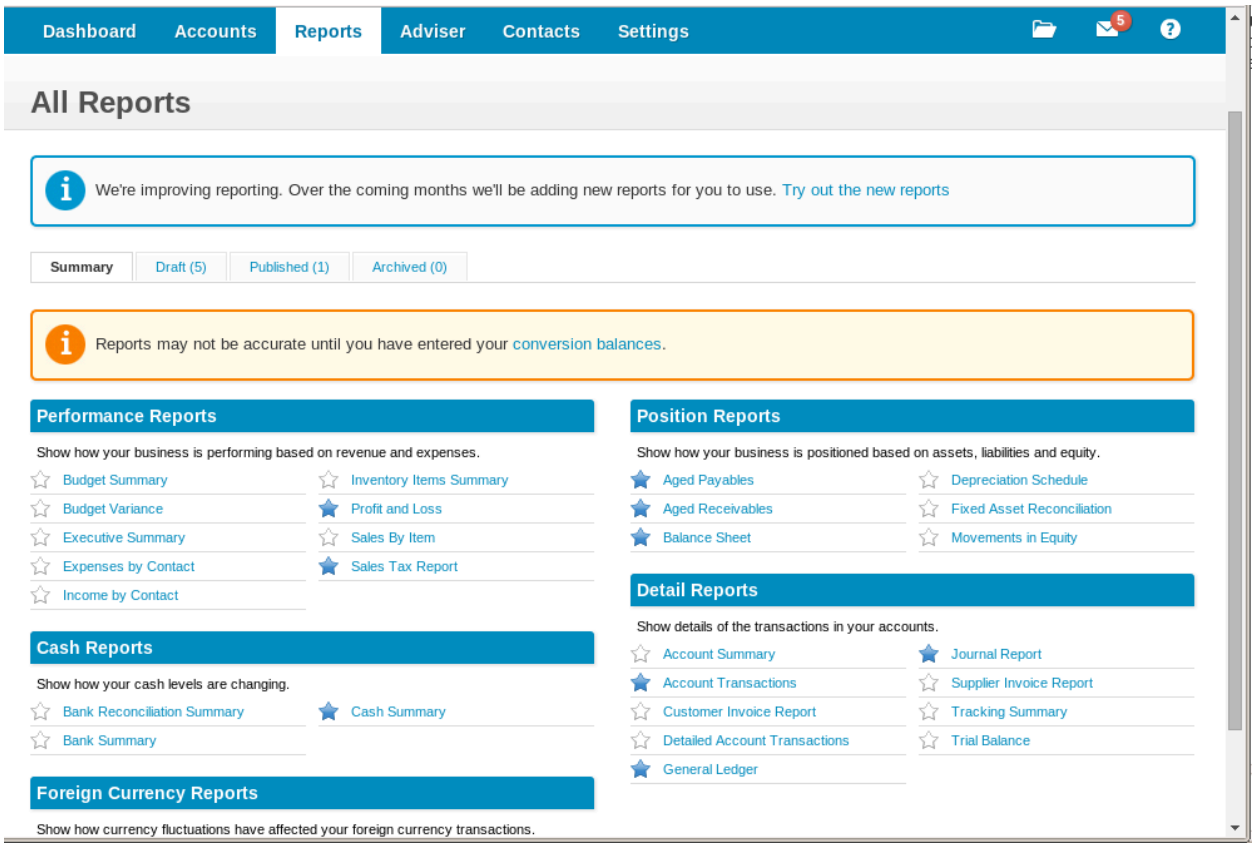
The number of menu items will drastically be reduced thanks to the dashboard. (kanban view of journals) Secondary features will be available from the dashboard.

The menu structure on the right: (new proposition sent by email from FP to CSN/QDP)

- Accounting
    - Dashboard
    - Journal Entries
- Customers
    - Sale Invoices
    - ~~Sale Refunds~~
    - Sales Receipt → should we keep it?
    - Customers
- Suppliers
    - Purchase Invoices
    - Purchase Refund
    - Purchase Receipts
    - Suppliers
- Statements
    - ~~Payment Followup~~ --> Customer Statements
    - ...
- Periodical Processing
    - Reconciliation
        - Manual
        - Automatic
- Reports
    - Profit And Loss
    - Balance Sheet
    - Aged Receivable
    - Taxes
    - All Reports (like in Xero → if a report is favourited, it's added in this menu)
- Configuration
    - Accounting
        - Fiscal Year → This form could be used to close a period / fiscal year

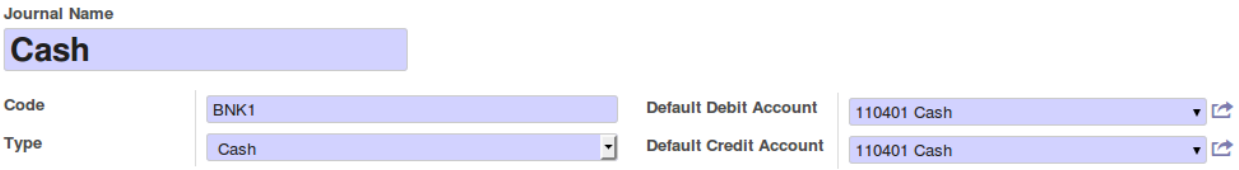
        - Accounts
        - Journals
        - ~~Bank Accounts → in technical feature for import only~~
        - ~~Models → model of entries should be defined where we use it.~~
    - Taxes
        - Taxes Definition
        - Fiscal Positions
    - Follow-Ups
        - Payment Terms
        - Followup levels

The “All reports” menu shows all reports and you can favourite some to make them appear in the menu.



TODO: mockup for this feature (and how we handle custom report definition)

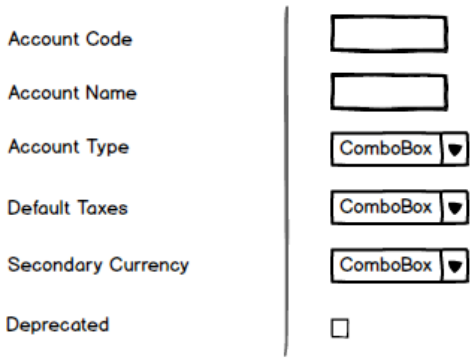
Journal form view (DONE)



Todo: improve this mockup:

- Code → Short Name on Reports / default prefix for sequence
- Explain the Type field (is there a tooltip?)
- Explain the link between journal and payment type, it's not usual.

Account form view (DONE)



[MGE] Assets & Revenue Recognition (done, to be merged)

Assets + New API Porting (done)

The asset category should appear conditionally according to the product. (add a many2one asset category on product.template object)

When setting an asset category, the account should change accordingly on the invoice (and when used in the PO).

On an asset, add a way to disengage the asset (e.g. you sold it).

The configuration of the accounts on products are not good:

- v8
    - No asset: income on product
    - Asset: income on asset and asset account on product

- Proposition for v9:
    - No asset: income on product
    - Asset: income on product (and other asset accounts on asset)

Port Asset to new API.

Revenue Recognitions (done)

Spec: ask mge

Recurring revenue dashboard (done)

A clone of baremetrics.io, to track recurring revenues.

Fiscal position form view (DONE)

add screenshot

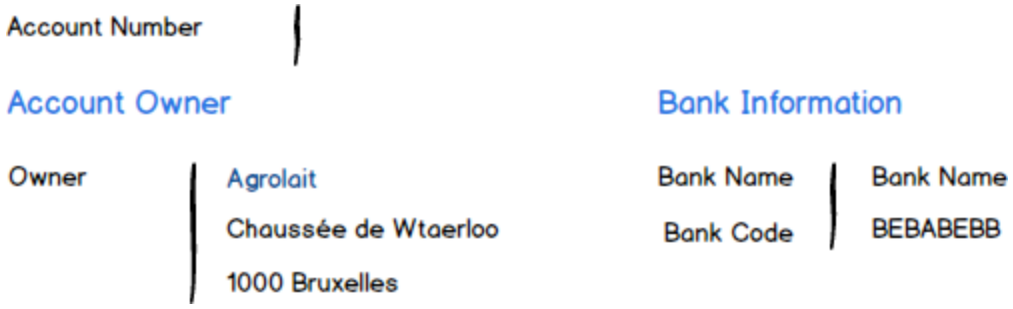
Invoice Report (DONE, to check)

- Allow to display shipping address on invoices
- When a fiscal position is set, we should display a related text on the invoice (a law requirement in some countries for some fiscal positions)
- Allow to display the sale date (that can be different from the invoice date) → Someone said it's mandatory in France, but I am not sure about it. → to be validated.

Remove All EDI stuff (done)

- We should remove or review EDI stuff. (should we keep it?) → Is anyone using it?

Bank Account form view (done, to improve)



TODO:

- Add a Type field: IBAN, Savings Account, ...
- Preload the list of banks?

Usability

User On Boarding Flow (DONE)

In summary, we have two WOW effects possible in just a few step:

Scenario 1:

1. Free Trial
2. Create Invoice
3. Send by Post (*)
4. (receive it in your mailbox 3 days after)

Scenario 2:

1. Free Trial
2. Sync With Bank (or Import a bank Statement)
3. Data are all filled and stuff configured:
    a. bank account created
    b. bank created as a contact
    c. bank journal created
    d. journal items available

So, after the free trial (or installation of the DB), the user arrives in the kanban dashboard with a few journal created (sales, purchases, bank, cash). We need a primary button on these journals and depending on the button he choose, he goes the Scenario 1 or Scenario 2.

(*) It may not be clear for the user what's the value of sending invoices by post automatically. It would be good to add a TIP for the first time, explaining what the wizard will do and why it's great! → In our company, the accountant send invoices once a week (because she prefers to print and post by batches). This means we can deliver invoices 4 days faster on average (3.5 due to the batch and 1 due to international letters when it's printed+posted in the right country directly) Since our average customer payment term is 30 days, by sending invoices 4 days earlier, we can have 13% of our monthly turnover in our bank account.

TODO: We need to do an on-boarding flow like this one:
https://fr.slideshare.net/secret/upjRZ9TgRHnSFX

Planner (In Progress, LLE)

The planner aims to guide new users through the initial configuration of Odoo as a day-to-day accounting solution. It's also an introduction to the basic principles of accounting for beginners (as there is a minimal level of knowledge required to use the module efficiently)

Mockup: http://qf6iua.axshare.com/accounting.html (work in progress)

Review all form / list views (DONE)

Once everything is finished, review all form / list views.

Phase 2: localization

1) we port all existing l10n modules to our new accounting models
2) first countries: France, Belgium, US
    a) belgium: nothing more to add?
    b) france: discuss with community
    c) us: see below + bank web services

Switzerland

need one extra field on the account.move.line to store the bvr reference (or need a way to create journal entries with "invoice bvr + a sequence"... chosen on the payment term?) (they don't want it on the general ledger for example...)
=> add a boolean on payment term line

France

https://www.odoo.com/apps/modules/8.0/l10n_fr_fec/ → this seems to be a legal requirements

US

Check writing (done, to be merged)

Working v7 branch (not US-specific! except report format) available at:
https://code.launchpad.net/~openerp-dev/openobject-addons/trunk-account_check_writing_jam

Improvements are:

- ~~Check's form view looks more like a real check~~
- Shows Supplier's invoice number
- Optionally spills over to multiple pages. You can pay more than 10 invoice lines.
- Optionally displays credits that are being applied with this check.
- Suppresses invoices that are not part of this check payment.
- Optionally works with pre-printed checks number or not.
- Optionally resets check sequence during a batch-print if needed.
- Optionally overwrite check numbers during a batch-print if needed.
- Works with standard US checks (top, middle, or bottom pane).

Options are to be found in the following locations:

- in Companies > Companies > Configuration tab

- in Accounting > Configuration > Journals > Journals > Advanced Settings tab
- in Accounting > Suppliers > Write Checks > Print button > Batch-print wizard

Chart of accounts (correction of default US templates)

- Integrate Avatax?

Cash discount

This is a common practice in the US to give an incentive to your customers to pay according to a certain delay:

- Get x% off if invoice is paid before {date}. Eg: Get 10% off if invoice is paid before December 15 even though the term on the invoice is “Pay by December 31”.

The way to handle that:

- We create a payment term of two lines:
  - 90% within 10 days
  - 10% within 2 months
- Once we receive the 90%, we:
  - reconcile the payment with the 90%
  - And, manually, if the payment of the 90% is in the maturity date, we:
    - reconcile with the 10%
    - put the write-off in a cash discount account

Would be great if the manual operation can be automated.

Deposit Ticket (done, to be merged)

You need to record a deposit ticket in Odoo because the day you do your bank statement reconciliation, you need to know that the 1 line on your statement that states $6000 is in fact the sum of 3 checks for $1000, $2000, and $3000 from different customers. It's not something you want to have to remember by heart.
So the situation is that you have 3 invoices paid in Odoo, and at the end of the month you receive a bank statement with only 1 line that says $6000 with just a note "checks deposit". You don't remember which checks you deposited, so you look at your deposit ticket.

Notes:

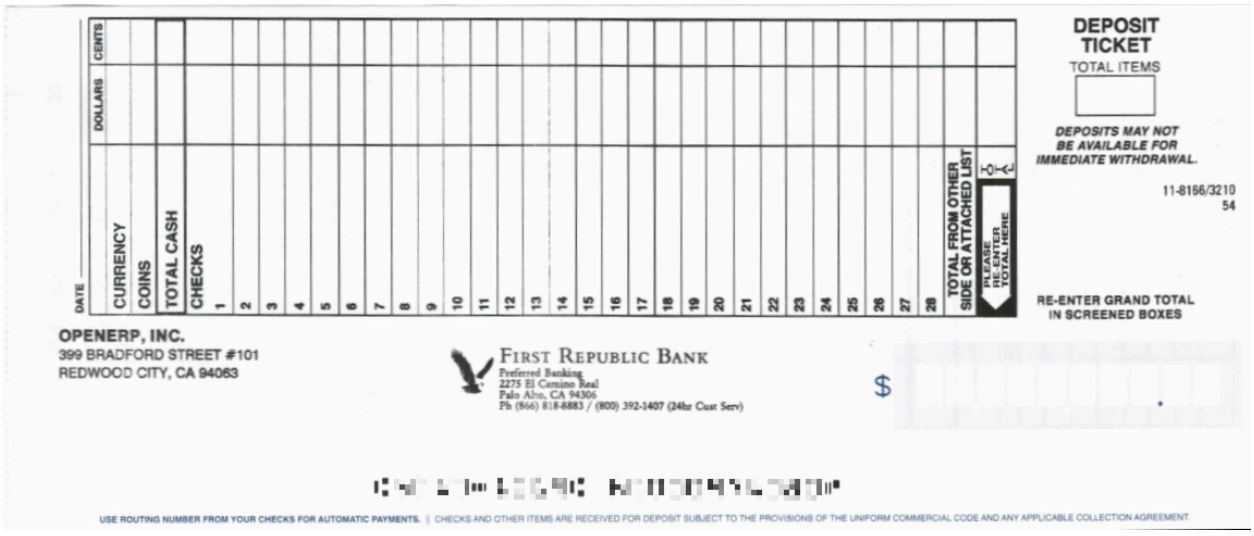
- Having a check printing feature doesn't help because we are talking about customer checks here, not checks that you print yourself.
- "Bank statement" is useless because it only indicates "checks deposit --- $6000", and it doesn't tell you which ones.

From what I have seen, Xero also has a deposit ticket feature, they call it batch deposit: https://help.xero.com/us/BatchDeposit
BTW, from my research New Zealand doesn't use checks so I don't think they are specialists.

In v8, what Odoo does well now with the new bank statement reconciliation tool is the ability to choose any customer invoice line when there is no partner on the bank account statement line. So the journal items are correctly created.
But we still have the problem of having to remember which checks we deposited in which batch.

This is the exact same problem with credit card payments. Imagine you settle a credit card batch every day made of 20 transactions. At the end of the month, you receive your bank statement with 30 lines (one per settlement). It will be impossible to use the bank statement reconciliation tool in Odoo because you will have to remember which transaction went into which batch and we are talking about 20*30=600 transactions to sort out. You cannot remember this by heart.

The purpose of a deposit ticket is to record the payments that constitute one bank deposit and therefore one line on a bank statement. These payments typically come from different customers. Only certain payment methods are relevant with deposit tickets: cash, checks, or credit cards. Not wire transfers.



Create a new object 'Deposit Ticket' to record a list of checks/cash payments that the user is going to deposit or a list of credit card payments that the user is going to clear. For customer payments received with checks or cash, the company will need to deposit those at the bank after they have been validated. They will be gathered into one deposit ticket that will include the information taken from the various customer payments and this deposit ticket will be imported on one bank statement.