

## Finish allocation processes

02/27/2023 08:11 PM - Alexander Blum

|                        |               |                        |             |
|------------------------|---------------|------------------------|-------------|
| <b>Status:</b>         | Neu           | <b>Start date:</b>     | 03/03/2023  |
| <b>Priority:</b>       | Normal        | <b>Due date:</b>       | 03/12/2023  |
| <b>Assignee:</b>       | Thomas Mielke | <b>% Done:</b>         | 10%         |
| <b>Category:</b>       |               | <b>Estimated time:</b> | 80.00 hours |
| <b>Target version:</b> |               | <b>Spent time:</b>     | 10.00 hours |

## Description

## Background

- Invoice.amount/shared\_amount are only helper attributes for testing the invoicing, should be replaced by  
UtilisationIndicators.invoice\_amount = UtilisationIndicators.administration\_fee + UtilisationIndicators.distribution\_amount
  - relationship see also [database diagram](#)
  - use erpservers> db-console for a tryton console to test the functions; maybe write a script with some objects initialized and import it there
- ```
>>> Tariff = pool.get('tariff_system.tariff')
>>> tariff = Tariff(1)
```

## Procedure

1. Write a [wizard](#) to trigger the allocation process
  - can be triggered by
    - cron job
    - manual [action](#)
      - in Declaration Menu (action in entry view or on [selected](#) items in list view)
      - in Utilization Menu (action in entry view or on [selected](#) items in list view). Maybe we could skip that, but probable it's a valid usecase, like:
        - if some utilization (tariff bound!) of a declaration should be payed for and some other has issues to be resolved first
        - or a multi day festival event and the utilisations should be split.
  - creates allocations with all corresponding utilisations connected, one allocation for each licensee
  - loops over different sets of declarations depending on where/how the action was triggered (algorithm should be generic, see next point confirm screen how this is done; not sure, how to handle the same action with different selections though, all other things should be similar to the existing example):
    - declarations list with no selection: all suitable declarations with suitable utilisations (default for the cron job e.g. each day)
    - declarations list with selection: all suitable declaration.utilisations for each selected suitable declaration
    - declaration entry: all suitable declaration.utilisations for this declaration (if suitable)
    - utilisations list with no selection: all suitable selected utilisations
    - utilisations list with selection: all suitable utilisations in list
    - utilisation entry: this utilisation (if suitable)
  - confirm screen (like in [fingerprint\\_merge](#), maybe a tree list with declarations -> utilisations)
    - list of definitely selected utilisations/declarations
    - info on deselection of non suitable utilisations/declarations
2. Write a dataset [triggering](#) the wizard for some Declarations
  - ensure, that non complete utilisation datasets for manual tests are still available
3. Implement the Tariff and TariffSystem formulas from [database diagram](#)
  - fomulas should have the exact same plain value parameters as stated in the tariff system (check the latest version), so that parametrized calling is possible via formula = getattr(tariff, 'method\_name'). they should **not** receive some tryton object, so they can be used (and tested) standalone

- each formula should be one **static** method

- TariffSystem parameters: tariff system version (dot replaced with underscore); like

```
@staticmethod
def formula_<tariffsystemversion>(...):
    [...]
    return invoice_amount
```

- Tariff parameters: tariff code (includes version already, dot replaced with underscore); one for each tariff; like

```
@staticmethod
def formula_<tariffcode>(...):
    [...]
    return base, relevance, adjustments
```

- maybe better change the tariff\_system.version.code, that it is save to use as function names in general and omit the replacements here

- add an **instance** method, to be able to easily fetch the corresponding formula from an instance

- TariffSystem

```
def get_formula(self):
    return getattr(self, f"formula_{self.version.replace('.', '_')}")
```

- Tariff

```
def get_formula(self):
    return getattr(self, f"formula_{self.code.replace('.', '_')}")
```

#### 4. Implement the **instance** methods

- params:

- sample: indicator set
  - e.g. 'estimated', 'confirmed', might depend on tariff
  - maybe also 'all' to calculate all indicator sets, but then the return value has to be e.g. a dict with sample name as key and the tuple (base, relevance, adjustments) as value
- update: write or just calculate the results

- maybe rename calculate\_invoice\_amount, as all values are returned, or maybe better split invoice amount calculation (only invoice\_amount returned) from calculating the distribution\_amount and administration\_fee (but not sure if this is needed by some other process)

- Allocation.calculate\_invoice\_amount(self, sample, update=False)

```
invoice_amount_sum = 0
distribution_amount_sum = 0
administration_fee_sum = 0
for utilisation in self.utilisations:
    invoice_amount, distribution_amount, administration_fee = \
        utilisation.calculate_invoice_amount(sample, update)
    invoice_amount_sum += invoice_amount
    distribution_amount_sum += distribution_amount
    administration_fee_sum += administration_fee
return invoice_amount, distribution_amount, administration_fee
```

- Utilisation.calculate\_invoice\_amount(self, sample, update=False)

```
utilisation = self
```

```
invoice_amount = self.tariff.calculate_invoice_amount(utilisation, sample, update)
return invoice_amount, distribution_amount, administration_fee
```

- `Tariff.calculate_invoice_amount(self, utilisation, sample, update=False)`

```
formula = self.get_formula()
```

```
context_indicators = getattr(utilisation.context, f'{sample}_indicators')
```

```
# generic way to get all and only indicators needed for all different formulas
```

```
import inspect # place import on top of file
```

```
formula_indicators = {
    field: getattr(context_indicators, field)
    for field in inspect.signature(formula).parameters
}
```

```
base, relevance, adjustments = formula(**formula_indicators):
```

```
invoice_amount, distribution_amount, administration_fee = \
    tariff_system.calculate_invoice_amount(base, relevance, adjustments)
```

```
if update:
```

```
    utilisation_indicators = getattr(utilisation, f'{sample}_indicators')
    utilisation_indicators.base = base
    utilisation_indicators.relevance = relevance
    utilisation_indicators.adjustments = adjustments
    utilisation_indicators.invoice_amount = invoice_amount
    utilisation_indicators.distribution_amount = distribution_amount
    utilisation_indicators.administration_fee = administration_fee
    utilisation_indicators.save()
```

```
return invoice_amount, distribution_amount, administration_fee
```

- `TariffSystem.calculate_invoice_amount(self, base, relevance, adjustments)`

```
formula = self.get_formula()
```

```
invoice_amount = formula(base, relevance, adjustments)
distribution_amount = invoice_amount * self.administration_share
administration_fee = invoice_amount - distribution_amount
# check if results add up
```

```
return invoice_amount, distribution_amount, administration_fee
```

5. Implement the same for `calculate_utilisation_indicators()`, maybe reuse in `calculate_invoice_amount()`

- `Utilisation.calculate_utilisation_indicators(self, sample, update)`
- `Tariff.calculate_utilisation_indicators(self, utilisation, sample, update)`
- best case: each [formula o node](#) with a custom method to use, and one method to integrate all of them (e.g. another `calculate_invoice_split()` function)

6. Implement a wizard to recalculate invoice amounts / utilisation indicators (prevent, if invoice is already issued)

7. Check the wizard to write invoices ([exists](#) already)

8. Write datasets to trigger the invoice action for the Allocations

- ensure, that some allocations are still left to invoice for manual tests

## Associated revisions

Revision 9c5fd1c7 - 03/06/2023 04:18 PM - Thomas Mielke

added wizard for allocation; ref #1140

renamed wizard 'allocate' to 'collect'; ref #1140

old wizard name: Allocate/utilisation.allocate      new: Collect/utilisation.allocation.collect  
old ModelView name: AllocateStart/utilisation.allocate.start      new: CollectStart/utilisation.allocation.collect.start

## History

---

### #1 - 02/27/2023 08:41 PM - Alexander Blum

- Description updated

### #2 - 02/27/2023 09:03 PM - Alexander Blum

- Description updated

### #3 - 02/27/2023 09:29 PM - Alexander Blum

- Description updated

### #4 - 02/27/2023 09:31 PM - Alexander Blum

- Description updated

### #5 - 02/27/2023 09:33 PM - Alexander Blum

- Description updated

### #6 - 02/27/2023 09:36 PM - Alexander Blum

- Description updated

### #7 - 02/27/2023 09:36 PM - Alexander Blum

- Description updated

### #8 - 02/27/2023 09:39 PM - Alexander Blum

For the development of the functions, the [tryton console](#) will be your friend:

```
$ docker compose run --rm --service-ports erpserver bash  
> db-console
```

### #9 - 02/27/2023 10:50 PM - Alexander Blum

- Description updated

### #10 - 02/27/2023 11:02 PM - Alexander Blum

- Description updated

### #11 - 02/28/2023 12:23 AM - Alexander Blum

- Description updated

### #12 - 02/28/2023 12:42 AM - Alexander Blum

- Description updated

**#13 - 02/28/2023 12:50 AM - Alexander Blum**

- Description updated

**#14 - 02/28/2023 12:55 AM - Alexander Blum**

- Description updated

**#15 - 02/28/2023 01:05 AM - Alexander Blum**

- Description updated

**#16 - 02/28/2023 01:14 AM - Alexander Blum**

- Description updated

**#17 - 02/28/2023 01:26 AM - Alexander Blum**

- Description updated

**#18 - 02/28/2023 01:28 AM - Alexander Blum**

- Description updated

**#19 - 02/28/2023 01:29 AM - Alexander Blum**

- Description updated

**#20 - 02/28/2023 01:34 AM - Alexander Blum**

- Description updated

**#21 - 02/28/2023 01:44 AM - Alexander Blum**

- Description updated

**#22 - 02/28/2023 01:51 AM - Alexander Blum**

- Description updated

**#23 - 03/03/2023 06:46 PM - Thomas Mielke**

- Due date set to 03/12/2023

- Start date set to 03/03/2023

- Estimated time set to 80.00

**#24 - 03/03/2023 07:32 PM - Thomas Mielke**

what do you mean by "dot replaced with underscore"?

**#25 - 03/03/2023 11:40 PM - Alexander Blum**

in our demo data the tariff\_system.code is e.g. "1.0". dots are not allowed in method names, so instead of "formula\_1.0()" sanitize the name to "formula\_1\_0()". same with tariff code, which is just "self.code + self.tariff\_system.code". or restrict tariff\_system.code and tariff.code to contain only chars allowed in function names.

**#26 - 03/03/2023 11:58 PM - Alexander Blum**

- Description updated

**#27 - 03/04/2023 12:02 AM - Alexander Blum**

- Description updated

- % Done changed from 0 to 10