# Numérotation des affaires et des documents commerciaux

Les affaires peuvent être numérotées selon un des modèles ci-après :

1. Numérotation séquentielle.
2. N° de l’année et numérotation séquentielle.
3. N° de l’année et numérotation séquentielle par année.
4. N° de client et l’une des variantes 1 à 3, avec séquence globale.
5. N° de client et l’une des variantes 1 à 3, avec séquence propre au client.

Afin de simplifier la production des numéros, on part de l’idée que toute séquence est liée à une clé. Le numéro final produit peut inclure la clé et un préfixe. Ainsi :

1. Clé vide, numéro.  
   0000**1**, 0000**2**, 0000**3**, …
2. Clé vide, préfixe « année » + numéro.  
   2010-000**1**, 2010-000**2** … 2010-000**4**, 2011-000**5**, 2011-000**6** …
3. Clé « année », préfixe « année » + numéro.  
   2010-000**1**, 2010-000**2** … 2010-000**4**, 2011-000**1**, 2011-000**2** …
4. Comme 1 à 3, avec un préfixe « client » supplémentaire.  
   1008/000**1**, 1008/000**2**, 1024/000**3**, 1008/000**4** …  
   1008/2010-000**1**, 1008/2010-000**2**, 1024/2011-000**3**, 1008/2011-000**4** …  
   1008/2010-000**1**, 1008/2010-000**2**, 1024/2011-000**1**, 1008/2011-000**2** …
5. Comme 1 à 3, avec une clé « client » et un préfixe « client » supplémentaire.  
   1008/000**1**, 1008/000**2**, 1024/000**1**, 1008/000**3** …  
   1008/2010-000**1**, 1008/2010-000**2**, 1024/2011-000**1**, 1008/2011-000**3** …  
   1008/2010-000**1**, 1008/2010-000**2**, 1024/2011-000**1**, 1008/2011-000**1** …

Détail d’implémentation : la classe Business.RefIdGenerator génère des numéros séquentiels uniques, propres au générateur défini par un nom. Celui-ci résulte de la concaténation du DRUID de l’entité (par ex. AffairEntity) et de la clé optionnelle.

Les documents commerciaux peuvent reprendre le n° d’affaire comme préfixe ou l’intégrer à leur clé. Ils peuvent aussi inclure un code qui représente le type de document considéré (offre, bon pour commande, confirmation de commande, bulletin de livraison, facture, autre), sous la forme d’un préfixe ou d’une clé.

Détail d’implémentation : si le numéro de document dépend du n° d’affaire, le nombre de documents au sein d’une séquence restera faible. Il n’est dès lors techniquement pas utile de définir un générateur, car l’unicité des numéros au sein d’une affaire peut être garantie par un verrou.

Exemples de n° de documents qui peuvent être produits avec ce schéma (10 = offre, 50 = facture) :

* aaaa.10.01, aaaa.10.02 🡪 variantes 01 et 02 de l’offre
* aaaa.50.01, aaaa.50.01.1, aaaa.50.01.2 🡪 facture initiale 01, 1er et 2ème rappels de la facture
* aaaa.5010, aaaa.5011, aaaa.5012 🡪 facture initiale 01, 1er et 2ème rappels de la facture

L’implémentation actuelle se trouve dans la méthode FormattedIdGenerator.AssignIds… Le format est géré par la classe FormatterHelper (exemples : yy, yyyy, ##, n, nn, nnn, …) et les formats avancés sont fournis au travers de classes qui implémentent IFormatTokenFormatter, par exemple #doc() est implémenté par DocumentInAffairReferenceNumberFormatter.

# Documents commerciaux

Un document commercial se décompose en plusieurs blocs :

* Les métadonnées du document (type de document, date d’établissement, libellé, n°, auteur, etc.).
* Les informations générales liées au document commercial (client, adresses, comptabilisation).
* Les lignes d’articles y compris titre, textes, sous-totaux, rabais et groupes.
* Le récapitulatif TVA (une ligne par code/taux).
* Le grand total (HT, TTC, prix arrêté[[1]](#footnote-1)).

En outre, dans le cas d’une facture, des informations de paiement (BillingDetails) viennent compléter le document à proprement dit (BVR, montant dû, date d’échéance, encaissement, etc.).

# Documents commerciaux et rappels

Un rappel est un document commercial similaire à une *facture*. Il fait référence à une facture de base, mais ne reprend pas le détail des articles.

Un rappel est donc une facture qui comprend les informations suivantes :

* Texte : libellé, n°, date d’émission de la facture originale.
* Texte : récapitulatif des montants facturés (date d’échéance, HT, TTC et total TVA sans détail).
* Texte : liste des encaissements enregistrés pour la facture (s’il y en a), avec date et montant.
* Optionnellement, une ligne d’article avec frais de rappels (montant HT, TTC et TVA).
* Optionnellement, des lignes avec titre et textes saisis par l’utilisateur.
* Optionnellement, un escompte accordé (faut-il prévoir un article correspondant ?).
* Le récapitulatif TVA (une ligne par code/taux).
* Le montant encore dû qui apparaît en lieu et place du grand total.
* Les informations de paiement relatives au rappel.

La comptabilisation d’une facture de type rappel ne reprend que les lignes d’articles du rappelhe (dans ce cas, on ne passera en comptabilité que les frais de rappels, la TVA associée à ces frais et un éventuel escompte).

# Documents commerciaux et quantités

Le prix unitaire se calcule en fonction de la **quantité commandée**.

* L’*offre* spécifie la **quantité commandée**, éventuellement déjà des informations de disponibilité ou de date de livraison estimée, au moyen de **quantités en suspens** avec des dates échelonnées.  
  Les quantités en suspens décrivent simplement quand la quantité commandée sera livrée et elles n’interviennent pas dans les calculs.
* Le *bon pour commande* peut être traité soit comme l’offre (sans aucune information supplémentaire liée à des dates de livraison, par exemple), soit comme la confirmation de commande (cf. ci-après).
* La *confirmation de commande* spécifie la **quantité commandée**, les **quantités confirmées** (avec des dates de livraison échelonnées) et la **quantité offerte** (par ex. parce que le client commande 19 pièces et qu’un lot en compte 24, qu’on ne souhaite pas le fractionner et qu’on préfère livrer les 24, en offrant les 5 pièces).
* Le *bulletin de livraison* spécifie la **quantité livrée** et les **quantités encore en suspens** (avec des dates de livraison échelonnées, voire une mention telle que « épuisé »). Aucun prix n’est indiqué.
* La *facture* spécifie la **quantité commandée** (qui détermine le prix unitaire) et la **quantité facturée**. Elle peut rappeler les quantités encore en suspens, déjà facturées ou offertes, à titre d’information.

Pour résumer, les types de quantités suivants doivent pouvoir être différenciés :

* Quantité commandée : ArticleQuantityType.Ordered.
* Quantité facturée : ArticleQuantityType.Billed.
* Quantité confirmée et/ou en suspens : ArticleQuantityType.Delayed.
* Quantité en suspens avec date non confirmée : ArticleQuantityType.Expected.
* Quantité livrée : ArticleQuantityType.Shipped.
* Quantité livrée précédemment : ArticleQuantityType.ShippedPreviously.
* Quantité à titre d’information (par ex. quantité offerte) : ArticleQuantityType.Information.

Le passage d’un type de document à l’autre est géré par la classe AffairActions, via un workflow. Contraire­ment à Crésus Facturation 9, chaque document commercial est une copie complète et indépendante. On copie dès lors l’offre pour en faire un bon pour commande, puis on copie le bon pour commande pour produire la confirmation de commande, etc.

### Établissement de l’offre

Au moment de l’établissement de l’offre, l’interface ne doit pas présenter le détail des quantités. Il faut prévoir une saisie simplifiée directement au niveau de la ligne d’article de la quantité commandée (Ordered).

À l’avenir, il faudra aussi pouvoir spécifier une ou plusieurs quantités en suspens (Expected) avec des dates estimées. Gérer les cas suivants :

* La quantité commandée dépasse les quantités en suspens (ou il n’y a aucune quantité en suspens) : une partie de/toute la commande sera honorée immédiatement.  
  Afficher les quantités livrées de suite s’il y a des quantités en suspens.
* La quantité commandée est plus faible que les quantités en suspens : les quantités seront probablement ajustées au moment de la confirmation de commande.  
  Afficher un avertissement pour attirer l’attention sur ce fait.

La saisie des quantités en suspens pourra se faire dans une tuile séparée, détaillée.

### Édition de la confirmation de commande (éventuellement bon pour commande)

À ce stade, les articles ne peuvent plus être modifiés. Seules les quantités doivent pouvoir être modifiées : la quantité commandée (Ordered) et une ou plusieurs quantités confirmées/en suspens (Delayed), avec les dates correspondantes. S’il y avait des quantités en suspens dans l’offre (Expected), elles doivent être converties en quantités confirmées/en suspens (Delayed).

En principe, la somme des quantités confirmées/en suspens correspond à chaque quantité commandée. Dans un document simplifié, les quantités confirmées/en suspens peuvent ne pas être spécifiées du tout.

La quantité commandée est éditée au niveau de la ligne d’article. Les autres quantités apparaissent sous la forme d’un tableau à *n* lignes. L’interface peut offrir des indications (par ex. total confirmé/en suspens pas égal au total commandé), comme cela est aussi le cas pour l’offre détaillée.

### Établissement d’un bulletin de livraison (BL)

Le bulletin de livraison doit permettre une saisie rapide des quantités livrées (Shipped), en repartant du dernier document produit. La quantité commandée (Ordered) n’est évidemment plus modifiable. Les quantités déjà livrées et décomptées sur des BL précédents (ShippedPreviously) sont maintenues à titre d’information. Les quantités en suspens correspondent au solde à livrer.

### Établissement d’une facture

La facture reprend soit ➀ l’entier des quantités de la confirmation de commande ou ➁ les quantités livrées d’un ou plusieurs bulletins de livraison. Les quantités sont soit facturées (Billed), soit en suspens (Delayed), soit présentes à titre d’information (Information). Pour le calcul du prix unitaire, la quantité commandée est reprise (Ordered), sans que cette information ne doive forcément apparaître dans le document produit.

L’interface permet de comparer aisément ce qui a été commandé, ce qui a été livré, ce qui a déjà été facturé et ce qui doit encore l’être. En particulier, si plusieurs factures doivent être émises, l’utilisateur doit conserver une vue d’ensemble pour éviter des erreurs (double facturation ou oublis).

# Documents commerciaux et lignes d’articles

Le document commercial comprend une collection ordonnée de lignes d’articles (nota bene : c’est un abus de langage, puisqu’il peut aussi s’agir d’autres éléments, tels que des textes, des sous-totaux, des rabais, etc.). Elles dérivent toutes de l’entité abstraite AbstractDocumentItem et peuvent être classées comme suit :

* Article.
* Titre ou texte.
* Sous-total.
* Rabais.
* Taxe (par ex. taxe sur le CO2, RPLP).
* Frais de rappel.
* Frais de port et d’emballage.
* Total.
* TVA.
* Grand total.

Remarques :

* Un groupe de lignes se termine toujours par un sous-total qui peut à son tour être suivi d’un ou plusieurs rabais, et d’un sous-total final.
* Les taxes et les frais sont représentés par des lignes d’articles standards, même si elles sont présentées séparément à l’utilisateur.
* Les lignes de TVA sont produites automatiquement par le calculateur de prix et viennent toujours se rajouter à la suite du grand total.
* Le grand total est unique et vient toujours après toutes les lignes que l’utilisateur peut éditer.
* Les lignes synthétisées par le calculateur de prix se reconnaissent à l’attribut AutoGenerated.

## Attributs

Chaque ligne d’article possède des attributs, définis dans l’énumération DocumentItemAttributes :

* None, pas d’attribut particulier.
* Hidden, ligne cachée (mais prise en compte par le calculateur de prix, cela peut provoquer des surprises chez l’utilisateur— voir s’il ne faut permettre de cacher que des titres, textes et totaux).
* AutoGenerated, ligne produite automatiquement (par le calculateur de prix).
* ProFormaOnly, ligne pro forma, sans incidence sur le prix (elle est ignorée dans les totaux faits par le calculateur de prix).

## Sous-totaux

Une ligne de sous-total dans le BusinessDocument produit potentiellement plusieurs lignes à l’écran ou sur le papier :

* Sous-total avant rabais (PrimaryPriceBeforeTax et PrimaryTax).
* Rabais (Discount).
* Sous-total après rabais (ResultingPriceBeforeTax et ResultingTax).

L’énumération PriceDisplayModes disponible dans le champ DisplayModes spécifie quelles informations doivent être rendues visibles :

* None, aucune information.
* PrimaryTotal, montre le sous-total avant rabais.
* ResultingTotal, montre le sous-total après rabais.
* Discount, montre le rabais.
* Tax, montre les détails de TVA.

## Groupes

Chaque ligne est liée à un groupe (identifié par son **group index**, stocké dans GroupIndex). Ceci permet une organisation visuelle et hiérarchique. L’index encode à la fois le n° du groupe et l’identité du groupe parent, en fonction du niveau d’imbrication (nommé **group level**, calculé à partir de l’index et disponible comme propriété GroupLevel dans chaque entité) :

* 00 correspond à un groupe spécial nommé **section de pied** ou **groupe racine**.
* 01, 02, 03 sont des index des groupes à la racine du document.
* 0101, 0102, 0103 sont des index des groupes contenus dans le groupe 01.
* etc.

Exemple : un index de groupe égal à 20401 (décomposé comme 02 04 01) correspond donc à l’index d’un groupe 01, contenu dans le groupe 04, contenu dans le groupe 02. Le groupe est de niveau 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N° de ligne | Group level | Group index | Type | Contenu |
| 1 | 1 | 01 | Titre | … |
| 2 | 2 | 0101 | Article A | 10.– |
| 3 | 2 | 0101 | Article B | 20.– |
| 4 | 2 | 0101 | Article C | 30.– |
| 5.1 | 2 | 0101 | Sous-total groupe 01 01 | 60.– |
| 5.2 | 2 | 0101 | Rabais | 10.– |
| 5.3 | 2 | 0101 | Sous-total récapitulatif 01 01 | 50.– |
| 6 | 2 | 0201 | Titre | … |
| 7 | 3 | 010201 | Article D | 25.– |
| 8 | 3 | 010201 | Sous-total groupe 01 02 01 | 25.– |
| 9 | 3 | 020201 | Titre | … |
| 10 | 3 | 020201 | Article E | 10.– |
| 11 | 3 | 020201 | Article F | 10.– |
| 12 | 3 | 020201 | Sous-total groupe 02 02 01 | 20.– |
| 13 | 2 | 0201 | Sous-total groupe 02 01 (2 sous-groupes) | 45.– |
| 14.1 | 1 | 01 | Sous-total groupe 01 (2 sous-groupes) | 95.– |
| 14.2 | 1 | 01 | Rabais | 5.– |
| 14.3 | 1 | 01 | Sous-total récapitulatif 01 | 90.– |
| 15 | 0 | 00 | Article Frais | 10.– |
| 16 | 0 | 00 | Sous-total (total final avant TVA) | 100.– |
| 17 | 0 | 00 | TVA | 8.– |
| 18 | 0 | 00 | Grand total | 108.– |

1 Titre

2 A 10.–

3 B 20.–

4 C 30.–

5.2 Rabais -10.–

5.3 Sous-total après rabais 50.–

6 Titre

7 D 25.–

8 Sous-total 25.–

9 Titre

10 E 10.–

11 F 10.–

12 Sous-total 20.–

13 Sous-total 45.–

14.1 Sous-total 95.–

14.2 Rabais -5.–

15 Frais 10.–

16 Total HT 100.–

17 TVA @ 8.0% 8.–

**18 Grand total TTC 108.–**

Dans le cas d’une facture où les montants sont TTC dans les lignes d’articles, le pied de la facture se transforme en ce qui suit :

14.1 Sous-total 103.–

14.2 Rabais -5.–

15 Frais 10.–

**16 Total TTC 108.–**

17 TVA @ 8.0% incluse 8.–

Il suffit donc de cacher le grand total dans un tel cas, à moins qu’on ne souhaite forcer un total arrêté.

# Interaction pour la saisie d’articles

Considérations générales :

* Les groupes doivent être représentés visuellement et clairement identifiables.
* L’ajout d’un élément doit se faire en un clic unique : il faudrait des commandes explicites pour permettre d’ajouter groupe, sous-groupe, titre, texte, article, rabais, taxe et frais.
* Les éléments groupe, titre, texte, article et rabais s’ajoutent dans le groupe actif.
* L’élément groupe s’ajoute après le groupe actif.
* Les éléments de sous-total sont ajoutés automatiquement et peuvent être cachés par l’utilisateur.
* Les éléments de sous-total peuvent être ajoutés à la main pour introduire des cascades de rabais.
* Les éléments taxe et frais s’ajoutent à la fin, dans une section de pied.
* Les éléments groupe, titre, texte et article peuvent être déplacés librement, même entre différents groupes. Ils ne peuvent pas être placés dans la section de pied.
* Les éléments rabais, taxe et frais peuvent être déplacés les uns par rapport aux autres.
* Les groupes doivent pouvoir être fusionnés ou scindés.
* Quand un groupe contient un unique sous-groupe, il faudrait cacher le sous-total du sous-groupe pour éviter des doublons dans la présentation de la facture.

Illustration (ligne 5) :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N° de ligne | Group level | Group index | Type | Contenu |
| 1 | 1 | 01 | Titre | … |
| 2 | 2 | 0101 | Article A | 10.– |
| 3 | 2 | 0101 | Article B | 20.– |
| 4 | 2 | 0101 | Article C | 30.– |
| 5 | 2 | 0101 | Sous-total groupe 0101 | 60.– |
| 6 | 1 | 01 | Sous-total groupe 01 | 60.– |
| 7 | 0 | 00 | Frais | 10.– |
| 8 | 0 | 00 | Total TTC | 70.– |
| 9 | 0 | 00 | TVA incluse | 5.19 |
| 10 | 0 | 00 | Grand total | 70.– |

Nota bene : dans cet exemple, les montants sont TTC, il y a donc pas lieu d’afficher le grand total et l’indication de TVA est claire sur le fait qu’il s’agit d’une taxe sur la valeur ajoutée *incluse*.

* Les éléments de TVA et le grand total se calculent automatiquement. L’utilisateur ne peut pas intervenir sur les éléments de TVA.
* Le grand total peut être manuellement forcé à un **total arrêté**.

# Documents commerciaux et calculateur de prix

Que la facture contienne des lignes d’articles HT ou TTC n’influe aucunement sur les données stockées. C’est surtout une question d’affichage, dans la mesure où tous les éléments contiennent à la fois les montants HT et TTC. Dans le cas d’un rabais (ou du total arrêté), l’utilisateur doit spécifier s’il s’agit d’un montant HT ou TTC.

Le calculateur de prix est implémenté par la classe DocumentPriceCalculator, mais il n’est jamais appelé directement. Il doit être passé en argument à PriceCalculator.UpdatePrices, et utilisé au sein d’un bloc using. Il est par exemple invoqué automatiquement lorsque BusinessDocumentEntity est modifiée, grâce à une règle de la *business logic* liée.

## Algorithme utilisé par le calculateur de prix

Le calculateur de prix travaille en plusieurs passes :

1. Il trie les lignes du document commercial.
   1. Les lignes sont regroupées par groupe et leur ordre respectif n’est pas modifié.  
      01:a/0101:b/01:c/0101:d/02:e/0201:f/03:g 🡪 01:a/0101:b/0101:d/01:c/0201:f/02:e/03:g
   2. Les sous-totaux sont placés en fin de chaque groupe.
   3. Le groupe zéro (section de pied) est trié de manière spécifique, dans l’ordre suivant :  
      sous-groupes, articles, sous-totaux, lignes de TVA, grand total puis textes éventuels.
   4. Au besoin, le calculateur rajoute ou supprime des sous-totaux, là où cela est nécessaire.  
      🡪 Un groupe contenant des articles ou des sous-groupes non vides a forcément un sous-total.  
      🡪 Un groupe ne contenant aucun montant n’a pas de sous-total.
   5. Pour terminer, le calculateur corrige la section de pied en s’assurant qu’il y ait au moins un sous-total ou en supprimant tous les sous-totaux si le document ne contient aucun article.
2. Il calcule les prix et met à jour le contenu des lignes.  
   Pour cela, il appelle la méthode Process de chaque ligne. Les lignes de type article et sous-total créent un calculateur spécifique et appellent à leur tour DocumentPriceCalculator.Process (il existe une implémentation par type de ligne). Ce processus produit un graphe de calculateurs de groupes qui produisent en fin de compte un total de TVA sur les articles avec rabais possible, un total de TVA sur les articles sans rabais possible et un grand total.
3. Il accumule la TVA et produit les lignes de taxes liées à la TVA, ainsi que le grand total, qu’il place à la fin du groupe racine, avant d’éventuels textes.
4. Il calcule les prix définitifs à utiliser pour la comptabilisation de chaque ligne d’article.

Les lignes d’articles *pro forma uniquement* ne sont pas totalisées. Le calculateur se contente de mettre à jour les prix calculés au sein de la ligne concernée, mais traite sinon l’entier de la ligne comme du texte informationnel.

## Calculateurs de prix pour les lignes en général

Les calculateurs de prix spécifiques aux lignes, dérivant de AbstractItemPriceCalculator, sont définis. Leur responsabilité est de calculer le prix définitif (ApplyFinalPriceAdjustment) :

* ArticleItemPriceCalculator 🡪 lié à une ligne d’article.
* SubTotalItemPriceCalculator 🡪 lié à un sous-total.
* GroupItemPriceCalculator 🡪 lié à un groupe.

## Calculateurs de prix pour les articles

La méthode DocumentPriceCalculator.Process(ArticleItemPriceCalculator) fait ceci, pour autant que l’article ne soit pas de type *pro forma uniquement* :

1. Si c’est le premier article rencontré, conserve la trace du calculateur du groupe en cours (s’il y en avait un), et crée un nouveau calculateur pour le groupe.
2. Ajoute dans la liste des calculateurs le calculateur d’article.
3. Calcule le prix de l’article.
4. Ajoute dans le groupe actif les éventuels calculateurs des sous-groupes accumulés précédemment.
5. Ajoute dans le groupe actif le calculateur d’article.

sinon (si c’est un article de type *pro forma uniquement*), il se contente de reculer le prix de l’article.

Le calcul du prix de l’article (ComputePrice) se déroule en plusieurs étapes :

1. Détermine le mode d’arrondi à utiliser.
2. Détermine les quantités commandée et réelle.  
   La quantité réelle correspond au nombre d’articles à facturer dans le document concerné.
3. Détermine le prix d’article qui sera appliqué (ArticlePriceEntity) en fonction de la quantité commandée, de la date, etc.
4. Calcule le prix unitaire HT. Si un prix a été forcé (BillingUnitPriceBeforeTax), c’est ce prix qui sert de base, sinon il est déterminé sur la base du prix d’article (qui peut à son tour nécessiter un calcul pour un article paramétré). Le résultat est arrondi selon le mode d’arrondi en vigueur.
5. Calcule le prix de ligne HT (produit du prix unitaire HT et de la quantité réelle).
6. Calcule le prix de ligne HT résultant, après rabais. Les rabais sont appliqués avec ou sans TVA.
7. Calcule la TVA du prix de ligne HT et du prix de ligne HT résultant.
8. Arrondit tous les montants en fonction de la monnaie utilisée.
9. Calcule le ou les taux de TVA appliqué(s) et arrondit ceux-ci de manière à avoir 6 chiffres après la virgule lors d’un affichage en pourcents (8.000000%).

En principe, le prix unitaire devrait être forcé dès que l’on copie un document commercial (par ex. une offre pour en faire un bon pour commande) ou après que le document ait été envoyé au client, pour éviter qu’une mise à jour du prix des articles ne modifie un document commercial existant.

## Calculateurs de prix pour les sous-totaux

La méthode DocumentPriceCalculator.Process(SubTotalItemPriceCalculator) fait ceci :

1. Ajoute dans la liste des calculateurs le calculateur de sous-total.
2. Ajoute dans le groupe actif les éventuels calculateurs des sous-groupes accumulés précédemment.
3. Calcule le sous-total en se basant sur le contenu du groupe en cours.
4. Crée un nouveau calculateur pour le groupe, ne contenant que le sous-total.  
   Note : ceci permet de cascader des sous-totaux au sein d’un même groupe, en ayant chaque fois un calculateur de groupe incluant le calculateur précédent, évitant ainsi d’additionner articles et sous-total, alors qu’il suffit de repartir du sous-total en cours.

Ainsi, le calculateur de prix produit un graphe de calculateurs spécifiques liés entre eux.

## Calculateurs de prix et TVA

La complexité des calculateurs spécifiques est liée à la gestion correcte de la TVA et de la ventilation des rabais. La TVA est représentée par la classe Tax qui stocke :

* Le total de la TVA.
* Le total HT sur lequel s’applique la TVA.
* Le détail des composants, sous forme d’une liste de TaxRateAmount, comprenant le montant de TVA et les code de TVA et taux de TVA correspondants.

En effet, un montant de TVA peut être constitué de plusieurs éléments. Par exemple, une prestation de 2000.– facturées à cheval entre 2010 et 2011 à laquelle s’ajoute la livraison pour 100.– de nourriture en 2011 produirait cette information :

* 1000.– @ TVA 7.6% 🡪 76.–
* 1000.– @ TVA 8.0% 🡪 80.–
* 100.– @ TVA 2.5% 🡪 2.50
* Total de TVA 158.50 (taux moyen de 7.5476%)

Il est primordial de conserver ces détails quand la TVA des divers articles est additionnée pour pouvoir produire un récapitulatif en pied de facture.

Quand des montants sont accumulés, il faut distinguer ceux qui peuvent être soumis à un rabais de ceux qui ne peuvent pas l’être. C’est ce que fait le GroupItemPriceCalculator.

Exemple : un groupe avec un total de 1000.– est composé de 800.– d’articles standards et de 200.– d’articles sur lesquels aucun rabais n’est possible. L’utilisateur décide d’accorder un rabais de 10% sur ce groupe. Au moment de la comptabilisation du chiffre d’affaires produit par les articles de la facture, le rabais est traité comme suit :

* 10% de rabais sur 1000.– correspond à 100.–
* Le rabais de 100.– ne s’applique qu’aux articles standards (800.–).
* Les articles standards subissent donc un rabais à hauteur de 12.5%.

Et bien sûr, si un groupe contient un groupe, et que le rabais s’applique à la facture en entier, la ventilation remonte récursivement, en ne s’appliquant que là où des rabais sont possibles. Et si le montant du rabais est spécifié TTC, il faut retrouver le rabais en %.

## TVA en fonction d’une date

La classe TaxCalculator est utilisée par le calculateur de prix des lignes d’articles, en spécifiant soit la date de la prestation, soit la durée (date de début et date de fin) de la prestation.

La méthode TaxCalculator.ComputerTax permet de calculer la TVA (plusieurs taux possibles) en fonction d’un montant HT et d’un code de TVA. Pour cela, les réglages liés à la TVA sont lus dans la base de données.

Afin d’accélérer les accès, un cache rattaché au DataContext est implémenté par le composant TaxContext, lequel conserve un tableau de toutes les définitions de TVA en provenance de la base de données.

# Syntaxe pour le formatage

Le formatage au moyen de TextFormatter reconnaît un certain nombre de commandes :

* « a », *mark*, « x », « y », « z », *clearToMarkIfEmpty* : si « z » est vide, tout ce qui suit *mark* ne sera pas inséré dans le texte.
* « a », *ignore*, « b » : l’élément *ignore* est simplement supprimé — utile uniquement pour l’algorithme interne de formatage, permettant de remplacer à la volée des éléments qui doivent être supprimés en fin de compte.
* « a », « x », *ifEmpty* : si « a » est vide, produit « x », sinon produit « a ».
* « a », « x », « y », *ifElseEmpty* : si « a » est vide, produit « x », sinon produit « a », « y ».
* « x », *format:#func* : applique la commande de formatage *func* à la valeur « x ».  
  Les commandes de formatage sont gérées par des classes implémentant IFormatTokenFormatter.

En outre, les préfixes et suffixes suivants sont reconnus, une fois que toutes les commandes ont été traitées :

* « x », « ~abc » : si « x » est vide, « abc » ne sera pas inséré dans le texte.
* « abc~ », « x » : si « x » est vide, « abc » ne sera pas inséré dans le texte.

Exemples :

* « Poids : », poids, « inconnu », « kg », *ifElseEmpty*  
  🡪 Poids : 10.5 kg  
  🡪 Poids : inconnu
* 25, *format:#string {0:0.0}%*  
  🡪25.0%  
  Le formatage est implémenté par StringFormatTokenFormatter qui accepte comme argument le même type de chaînes de formatage que la fonction string.Format.

# La liste de gauche (le browser)

La liste de gauche est gérée par le contrôleur BrowserViewController. Il est réalisé sous la forme d’un composant rattaché à CoreApp, créé au démarrage de l’application par le MainViewController.

* BrowserList représente la collection de lignes de la liste de gauche. La liste est liée au même data context que le contrôleur.
* BrowserListItem représente une ligne de la liste de gauche (entité et texte court associé).
* BrowserViewController.ItemCreator gère la création d’entités (après avoir passé par un éventuel contrôleur de création propre aux entités sélectionnées).
* BrowserViewController.BrowserNavigationPathElement représente un élément de chemin pour l’historique de navigation. La méthode Navigate permet de réactiver la liste de gauche dans un état mémorisé.

Les réglages et boutons d’action de la liste de gauche sont gérés par le contrôleur BrowserSettingsController, aussi créé par MainViewController. Le conteneur dans lequel les éléments d’interface sont construits est sous le contrôle de MainViewController et peut changer de taille et d’emplacement en fonction de BrowserSettingsMode.

# Ajouter une base de données dans le ruban

Pour permettre à l’utilisateur de sélectionne une nouvelle « base de données » (data set) dans le ruban, procéder comme suit :

* Pour une entité XxxEntity générant un schéma et réglée avec les attributs RAC (génère un repository, affichage individuel, création individuelle), les noms Xyz ci-après seront la version plurielle du nom Xxx.
* Créer une image avec **Pictogram**, à enregistrer dans « S:\Epsitec.Cresus\Cresus.Core\Images » avec le nom « Base.*Xyz*.icon » (par ex. « Base.Customers.icon »).  
  Les pages avec dimensions 20 × 20 et 31 × 31 doivent exister.  
  En outre, la version 31 × 31 doit exister aussi dans le style *page active*.
* Ajouter cette image au projet « Cresus.Core » : dans le dossier « Images » du Solution Explorer, ajouter le fichier existant et le configurer avec « Build Action = Embedded Resource ».
* Recompiler les assemblies concernées et relancer Designer.
* Dans Designer, définir une commande avec un nom du genre Base.ShowXyz, puis enregistrer.
* Dans le projet « Cresus.Core », dossier « Controllers », éditer la méthode RibbonViewController.GetDatabaseMenuCommands et y ajouter :  
  yield return Res.Commands.Base.ShowXyz;

Il faut ensuite implémenter au moins un *summary view controller* pour l’entité correspondante. Si celle-ci doit pouvoir être éditée, il faut aussi implémenter un *edition view controller*.

# Ajouter un projet dans la solution

Pour ajouter un projet lié à Crésus Core dans la solution :

* Clic droit sur la solution, « Add » et « New Project… ».
* Choisir « .Net Framework 4.0 » dans la combo et sélectionner « Class Library » dans la catégorie « Visual C# ». Nommer le projet Cresus.Core.Library.*Xyz* et valider.
* Supprimer le fichier « Class1.cs ».
* Copier les liens « AssemblyInfo.Solution.cs » et « Epsitec.Cresus.snk » d’un autre projet depuis le dossier « Properties » dans la racine du projet fraîchement créé.
* Dans les propriétés du projet :
  + « Application », changer le « Default namespace » en Epsitec.Cresus.Core.
  + « Signing », cocher « Sign the assembly » et sélectionner le fichier .snk copié précédemment.
* Ajouter les références à « Common », « Cresus.Core.Library », « Cresus.DataLayer », etc. en fonction des besoins, supprimer les références pas utiles.
* Déplacer « AssemblyInfo.Solution.cs » et « Epsitec.Cresus.snk » dans le dossier « Properties ».
* Éditer le fichier « AssemblyInfo.cs » pour ne conserver que les attributs suivants :
  + AssemblyTitle, nom du genre « Cresus.Core.Library.*Xyz* ».
  + AssemblyDescription, texte libre (souvent vide).
  + AssemblyProduct, toujours « Crésus ».
  + AssemblyCopyright, notice de copyright standard, « Copyright © 2011, EPSITEC SA, 1400 Yverdon-les-Bains, Switzerland »
* Ajouter dans le projet les dossiers « Resources », comprenant « Cresus.Core.Library.*Xyz* » contenant à son tour « SourceCode ».
* Dans Designer, créer un nouveau module de référence, avec le chemin d’accès absolu défini sur « S:\Epsitec.Cresus\Cresus.Core.Library.*Xyz*\Resources », le nom « Cresus.Core.Library.*Xyz* » et le namespace source « Epsitec.Cresus.Core », de type « Application ».
* Déverrouiller le module (Ctrl-L).
* Ouvrir les informations du module (bouton > en-dessus des boutons « Textes », « Légendes », …) et adapter le namespace pour les ressources en « «Epsitec.Cresus.Core.Library.*Xyz* » ; sélectionner l’option « Textes riches ».
* Enregistrer, quitter et relancer Designer, puis revenir à Visual Studio.
* Revenir à Visual Studio.
* Ajouter les fichiers « module.info » et tous les fichiers « \*.resource » (« Add » et « Existing Item ») dans « Cresus.Core.Library.Xyz ». Vérifier que « Build Action = Content » et « Copy to Output Directory = Copy if newer ».
* Ajouter les deux fichiers « Entitites.cs » et « Res.cs » dans le sous-dossier « SourceCode » quand ils auront été générés (celui des entités n’est créé que s’il existe au moins une entité dans les ressources). Ces fichiers ont un « Build Action = compile » réglé par défaut et il n’y a donc rien de plus à faire.
* Après avoir créé des ressources dans d’autres langues, il faudra encore ajouter dans le même dossier les fichiers « \*.resource » correspondants (comme ci-dessus).

# Entités à créer

Dans les réglages de l’entreprise :

* Générateur pour les affaires (…/…/Affair/A/yyyy-nnnn/yyyy/1000)
* Générateur pour les documents (…/…/DocumentMetadata/A/#doc()/ /0)

# Composants

## Les composants liés à l’interface graphique

Les composants liés à l’interface graphique dérivent de ViewControllerComponent. Ils sont créés par une *factory* implémentant IViewControllerComponentFactory (pour un composant de type T, c’est en principe une classe Factory dérivant de DefaultViewControllerComponentFactory<T>). L’initialisation de ces composants est gérée par le constructeur du DataViewOrchestrator.

Les composants suivants sont actuellement implémentés :

* DataViewController
* MainViewController
* MainWindowController
* RibbonViewController

Pour retrouver un composant, il faut appeler DataViewOrchestrator.GetComponent<T>.

## Les composants en général

La classe CoreComponentFactory est responsable de trouver et créer les composants (RegisterComponents), puis de les initialiser (SetupComponents). Des implémentations spécifiques sont utilisées pour les types de composants suivants :

* ViewControllerComponent 🡪 ViewControllerComponentFactory.
* CoreDataComponent 🡪 CoreDataComponentFactory.
* CoreAppComponent 🡪 CoreAppComponentFactory.

1. Le prix arrêté n’intervient pas dans le calcul du prix et dans la comptabilisation. Il sert uniquement de base à l’établissement des informations de paiement. À l’encaissement, la différence entre le total et le prix arrêté sera traité comme un escompte. [↑](#footnote-ref-1)