# Travaux pour DR dans Crésus Core

## Impression de documents

* Dans l’en-tête du document, il faut remplacer « tralala » par la localité, siège de l’entreprise (cf. réglages d’entreprise disponibles via la méthode BusinessContext.GetCachedBusinessSettings).
* Dans l’en-tête du document, ne pas écrire « CH-1023 Crissier » mais simplement « 1023 Crissier ».  
  Si le destinataire est à l’étranger, écrire « 35000 Rennes » et « France » sur la ligne en-dessous.  
  Je suppose que cette logique doit être adaptée directement dans l’entité décrivant l’adresse ?

## Interface utilisateur

* Réglages de l’entreprise > Rappels : ne pas afficher le code
* Réglages de l’entreprise > Rappels : associer une définition d’article avec le rappel.  
  L’article doit être de type ArticleType.Admin (ArticleDefinition > ArticleCategory > ArticleType).

# 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 et quantités

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

* L’*offre* spécifie la **quantité commandée**.
* 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 livrée : ArticleQuantityType.Shipped.
* Quantité livrée précédemment : ArticleQuantityType.ShippedPreviously.
* Quantité à titre d’information (par ex. quantité offerte) : ArticleQuantityType.Information.

### Établissement de l’offre

Au moment de l’établissement de l’offre, l’interface ne doit pas présenter le détail des quantités. Au contraire, il faut prévoir la saisie simple d’une quantité unique (la quantité commandée) directement au niveau de la ligne d’article. En interne, cela produit uniquement des quantités de type Ordered.

### 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.

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).

### 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).

# 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.

# 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 réglée avec les attributs RAC (génère un repository, affichage individuel, création individuelle), les noms Xyz ci-après seront la version plurielle du nom Xxx.
* Définir une commande avec un nom du genre Base.ShowXyz.
* 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 ».
* 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*.

# 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)