Présentation de la fonction "Edit Code" du RPG ILE

- Les articles -

Date de mise en ligne : dimanche 4 novembre 2007

Description :

Cet article présente la BIF (Built in function) %EDITC, et sa petite soeur %EDITW, et vous propose un programme de service vous permettant de l'utiliser dans des programmes autres que RPG ILE (Adelia, Cobol...).

Environnement iSeries

Cet article présente la BIF (Built in function) %EDITC, et sa petite soeur %EDITW, et vous propose un programme de service vous permettant de l'utiliser dans des programmes autres que RPG ILE (Adelia, Cobol...).

Que vous soyez un développeur Adelia, RPG ou Cobol, vous avez très certainement été amené, un jour ou l'autre, à écrire un algorithme de formatage de nombre, destiné à retranscrire, avec force séparateur décimal, séparateur de milliers et signe négatif, une valeur numérique dans un champ alphanumérique. Bref, cela revient à coder un équivalent des fameux « codes d'édition » (en anglais « edit code ») utilisés couramment dans les DSPF et PRTF.

Les cas d'utilisation sont nombreux, comme par exemple :

- pouvoir afficher alternativement - selon différents critères relatifs à des règles métiers - un libellé ou un montant, dans une même zone de type alphanumérique, que cette zone se situe sur un DSPF ou un PRTF,

- remplir la partie gauche d'un montant avec des symboles tels que l'astérisque (*), ce qui est très pratique notamment pour l'impression de lettres-chèques,

- afficher les montants selon les règles en vigueur dans différents pays (le point et la virgule n'ayant pas la même signification pour tout le monde), technique généralement placée sous le terme impropre de "localisation".

La fonction %EDITC est une BIF (Built In Fonction) du RPG ILE que je trouve tout simplement géniale, car elle décharge complètement le développeur de cette gymnastique algorithmique (au demeurant très intéressante à mettre au point, quand on a le temps...).

La puissance de %EDITC n'a d'égal que sa simplicité d'utilisation. Si vous prenez la peine de l'utiliser, elle vous fera économiser de précieuses heures de développement (et les lignes de code qui vont avec). Après vous avoir décrit son fonctionnement, et même si elle est spécifique au langage RPG ILE, je vous montrerai que vous pouvez l'utiliser sans trop de difficultés dans vos programmes Adelia, et très certainement aussi dans vos programmes Cobol.

Nous parlerons brièvement de la petite soeur de %EDITC, qui s'appelle %EDITW (pour « Edit Word » ou « mot d'édition »), et qui n'est pas mal non plus.

Pour rappel, un article de XDOCS400 vous fournit la liste des codes d'édition existants. Liste des codes d'édition en programmation RPG/400

Exemples d'utilisation de %EDITC :

- pour formater la variable numérique WORIG avec le code d'édition « A », et la stocker dans la variable alphanumérique WDEST :

```
WDEST = %EDITC(WORIG :'A')
```

-

pour formater la variable numérique WORIG avec le code d'édition « J », remplir la partie à gauche de la partie entière avec des astérisques, et la stocker dans la variable alphanumérique WDEST :

```
WDEST = %EDITC(WORIG :'J' :*ASTFILL)
```

La fonction %EDITC accepte trois paramètres qui sont :

- paramètre 1 (obligatoire) : toute variable numérique
- paramètre 2 (obligatoire) : les codes d'édition A à D, J à Q, X, Y, Z, et 1 à 9. Le code d'édition Y est un code spécifique aux dates, il ne fonctionnera correctement que sur des dates au format 6.0. Les codes 5 à 9 sont des codes d'édition personnalisables, dont la définition est accessible au moyen de la commande WRKEDTD.
- paramètre 3 (facultatif) : il peut prendre plusieurs valeurs qui sont :

— *ASTFILL : remplissage de la partie gauche de la zone résultat avec des astérisques (très pratique pour l'impression de lettres-chèques)
— *CURSYM : affiche à gauche du nombre formaté le symbole monétaire courant défini au niveau du système (ou $ si non défini)
— 'X' : affiche à gauche du nombre formaté le symbole placé entre apostrophes (X dans l'exemple ci-contre). On peut y placer un « E » pour « euro », un « £ » pour la livre-sterling, etc...

La fonction %EDITC n'offre pas que des avantages (ce serait trop beau). J'ai relevé 2 inconvénients que je vous détaille ci-dessous.

Le premier inconvénient que j'ai relevé, est que les paramètres 2 et 3 n'acceptent pas de variable, on ne peut leur transmettre que des constantes. Cela interdit du coup d'écrire ceci :

```
WDEST = %EDITC(WORIG:WEDITC:WCOMPL)
```

On verra que cela complique un peu les choses pour une utilisation dans d'autres langages comme Adelia par exemple, même si ce n'est pas rédhibitoire (cf. programme RPGLE joint à cet article).

Le second inconvénient que j'ai relevé concerne le format de la variable numérique à formater. S'il s'agit d'une variable numérique ayant un format de 15 chiffres dont 6 décimales, comme par exemple : -1525,128600 et si j'applique un code d'édition A, j'obtiens ceci : 1.525,128600CR on voit donc que la fonction %EDITC ne nous permet pas d'agir sur la partie décimale du nombre formaté. Si on avait souhaité obtenir un nombre arrondi à 2 décimales après la virgule, on devine qu'il va être un peu compliqué de développer un composant générique qui interagirait sur l'intégralité du nombre à formater.

Si on contourne le problème en travaillant sur des variables numériques contenant seulement 2 décimales, on peut néanmoins se faire piéger par un cas particulier tel que celui-ci :

```
/FREE
 WORIG = 9180607,00 ;
 WDEST = %EDITC(WORIG:'Z') ;
 /END-FREE
```

Dans l'exemple ci-dessus, et si on ne fait rien pour éviter ça, WDEST aboutira au résultat suivant : 918060700. Ce n'est peut être pas exactement ce que l'on souhaitait obtenir (ou peut être que si, à vous de voir selon vos besoins).

Le programme RPG ILE que je vous fournis en complément de cet article est un programme conçu pour servir d'interface entre la fonction %EDITC et des programmes écrits dans des langages autres que RPG ILE, tels que CLP, Adelia, RPG III ou COBOL.

Pour la petite histoire, j'avais d'abord écrit une première version de ce programme, sous la forme d'un programme classique, qui recevait 3 paramètres en entrée et retournait un 4ème paramètre en sortie :

1. - valeur d'origine numérique (format : 13, dont 2 décimales)
2. - code d'édition alphanumérique (format : 1 caractère)
3. - troisième paramètre de %EDITC (format : 8 caractères)
4. - valeur de destination alphanumérique (format : 20 caractères)

Peu satisfait de la manière dont j'avais écrit le programme initial, du fait des contraintes de %EDITC vues plus haut dans cet article, j'en avais parlé à Serge Gomes qui m'avait confirmé le fait qu'il n'y avait pas de solution simple pour contourner cette difficulté. Serge m'a cependant proposé de modifier le programme pour un faire un programme de service appelable au travers d'une fonction. J'ai trouvé l'approche intéressante, et j'ai profité de l'aubaine pour me familiariser avec les programmes de service, avec l'aide de Serge. Nos différents échanges nous ont permis d'aboutir au programme joint à cet article, que vous pouvez invoquer dans vos programmes RPG ILE au travers d'appels du genre :

```
VADES = FORMATC(VAORI:EDITC:PARM3)
```

ou encore (le 3ème paramètre étant optionnel) :

```
VADES = FORMATC(VAORI:EDITC)
```

Maintenant, je vous propose d'entrer plus en détail dans la mécanique du programme RPG ILE SGFORMAT qui encapsule la fonction FORMATC. Après avoir reçu les paramètres tels que la valeur d'origine, le code d'édition, et le symbole d'édition complémentaire, le programme effectue plusieurs actions :

- passage de la variable numérique reçue par une variable numérique intermédiaire en format 23.2. On occulte donc arbitrairement les décimales inférieures au centime dans la version du programme jointe à cet article, nous verrons plus loin comment cette limite pourrait être contournée.

- réception du code d'édition dont la valeur est contrôlée, et passée en majuscule si nécessaire.

- réception du symbole d'édition complémentaire : dans notre version du programme, nous avons prévu qu'il supporte les paramètres *ASTFILL et *CURSYM (vus précédemment), ainsi que les valeurs $, £ et E (en remplacement du vrai symbole Euros qui n'est pas supporté par l'émulateur 5250 Client Access.

Le programme utilise le code d'édition reçu pour débrancher vers la procédure de traitement du code d'édition correspondant, procédure dont le code est le suivant (extrait de la procédure FMTCODEA pris sur le code d'édition 'A') :

```
BEGSR FMTCODEA ;
 If W_Symbol = *blanks ;
 W_Retour = %EDITC(W_ValOri:'A') ;
 Elseif W_Symbol = '*ASTFILL' ;
 W_Retour = %EDITC(W_ValOri:'A':*ASTFILL) ;
 Elseif W_Symbol = '*CURSYM' ;
 W_Retour = %EDITC(W_ValOri:'A':*CURSYM) ;
 Else ;
 If W_Symbol = 'E' ;
 W_Retour = %EDITC(W_ValOri:'A':'E') ;
 Elseif W_Symbol = '£' ;
 W_Retour = %EDITC(W_ValOri:'A':'£') ;
 Elseif W_Symbol = '$' ;
 W_Retour = %EDITC(W_ValOri:'A':'$') ;
 Else ;
 // autres cas de figures ignorés
 W_Retour = %EDITC(W_ValOri:'A') ;
 Endif ;
 Endif ;
 ENDSR ;
```

Vous l'avez peut être deviné avant même de lire l'intégralité du code, mais les limites de la fonction %EDITC m'ont obligé à faire une sérieuse entorse à un de mes principes qui est d'éviter toute duplication de code. En effet, comme la fonction %EDITC n'accepte pas de variable sur les 2ème et 3ème paramètres, j'ai dû créer une procédure de formatage pour chaque code d'édition, contenant un code identique (hormis le code d'édition lui-même). Ca m'a fendu le coeur de faire ça, je ne sais pas si je vais m'en remettre... snif ;)

Quelques codes d'édition ont été traités de manière spécifique, il s'agit des codes d'édition X, Y et Z.

Le code d'édition Y a fait l'objet d'un traitement particulier, visant à gérer des dates en 6.0 et en 8.0. Pour les dates en 8.0, la fonction %EDITC est inutilisable, aussi nous avons feinté en utilisant à la place la fonction %EDITW. Le symbole d'édition peut être laissé à blanc (dans ce cas on considère qu'il s'agit d'un formatage de type *DMY), ou il peut recevoir les valeurs *DMY (ou *JMA qui donne le même résultat), ou encore *YMD (ou *AMJ qui donne le même résultat).

Les codes d'édition 5 à 9, qui sont très spécifiques et personnalisables sur chaque AS/400 (cf. commande WRKEDTD) ne sont pas pris en compte. A vous de voir si vous voulez les intégrer et comment.

Je vois 2 avantages à utiliser ce programme plutôt que de réinventer la roue en codant mon propre algorithme de formatage :

1 - la fonction %EDITC est implémentée à un niveau plus bas de l'OS/400 qu'un programme RPG, sa rapidité d'exécution est donc supérieure à celle d'un programme RPG équivalent. Dans une utilisation au sein d'un programme interactif avec sous-fichier par exemple, toute solution permettant d'améliorer les performances est bienvenue.

2 - la fonction %EDITC permet de couvrir l'ensemble des codes d'édition disponibles sur l'OS/400, il serait vraiment dommage de s'en priver et de réinventer la roue.

Petite recommandation pour les développeurs Adelia : si vous décidez d'utiliser ce programme dans vos programmes Adelia, je vous recommande de l'intégrer dans une règle de gestion de type « paramètre », pour une réutilisation plus aisée.

Pour finir, petite précision concernant le fichier texte (zippé) joint à cet article. Il contient le source du programme de service SGFORMAT, le source du prototype d'interface, le code source du linkage, plus 2 exemples de programmes de tests, le premier pouvant vous servir de prototype pour constituer votre programme RPG d'appel si vous voulez utiliser la fonction FORMATC dans des programmes Adelia, Cobol ou RPG III. Le second programme effectuant une batterie de tests (avec dump) de la fonction EDITC, il constitue un bon exemple pour tester l'utilisation de cette fonction au travers de programmes RPG ILE. Avant la compilation des différents éléments, penser à effectuer un remplacement du mot clé &LIB par votre propre bibliothèque de compilation.

On a évoqué plus haut dans cet article le fait que le programme joint à cet article formate les variables avec 2 décimales systématiquement. Il serait envisageable de développer un programme qui viendrait s'intercaler entre la fonction EDITC et le programme appelant. Ce programme recevrait, en plus des paramètres de la fonction EDITC, un paramètre définissant le nombre de décimales souhaité. A partir de ce nombre de décimales, le programme intermédiaire appellerait non pas la fontion EDITC, mais une fonction EDITC0, ou EDITC1, ou EDITC2, ou... je sais c'est moche, cette solution obligerait à créer 10 fonctions quasi identiques (EDITC0 à EDITC9). C'est la raison pour laquelle Serge Gomes est en train de réfléchir à l'écriture d'un composant RPG utilisant la bif %EDITC selon une approche quelque peu différente. Vous entendrez donc parler de %EDITC très prochainement sur XDOCS.