The REBOL Documentation Project

-- FR - Documentation REBOL - Divers --

Divers

"Copy or not copy", telle est la question.

Philippe Le Goff

Première publication : 30 avril 2006, et mis en ligne le dimanche 30 avril 2006

Carl Sassenrath, CTO REBOL Technologies 9-Apr-2006 18:03 GMT Article #0006

Ok, ce sujet est profond et vaste, mais je veux vous faire réfléchir là dessus ...

Un problème que nous devons régler très bientôt est celui rencontré lorsqu'on copie ou lorsqu'on ne copie pas des séries REBOL (strings, blocks, etc.). Ce problème a un impact toutes les versions de REBOL et est important parce que les règles pour copier/ne-pas-copier doivent être fortement présentes dans votre esprit, tandis que vous programmez en REBOL.

Par exemple, voici le simple cas, où vous utilisez des chaînes de caractères littérales en REBOL, comme dans une fonction :

```
fun: func [str] [append "example " str]
```

La plupart d'entre vous reconnaisse ce problème. La première fois, nous appelons la fonction :

```
print fun "test"
example test
```

et la seconde fois :

```
print fun "this"
example testthis
```

La chaîne littérale a été modifiée. Pour prévenir cela, vous deviez écrire :

```
fun: func [str] [append copy "example " str]
```

C'est l'une des premières choses qu'un débutant apprend - et, habituellement, en s'étant fait avoir.

Mais, le problème devient vite vraiment important rapidement rapidement parce qu'il s'applique à toutes les séries de REBOL (series !) soit les blocs, des parens, des noms de fichier, des URLs, etc...

Par exemple, quand vous créez un objet de cette manière :

```
obj-body: [a: 10 b: [print a]]
obj1: make object! obj-body
obj2: make object! obj-body
```

Vous percevez notre problème si le bloc obj-body doit être copié avant qu'il soit lié (binding des variables attributs de l'objet). Le "obj-body" original est modifié par la fonction make. Si vous ne savez pas cela, vous pourriez être étonné. Surprise !

La même chose peut être dite à propos des spécifications ou du corps de fonctions, etc.

De sorte que ce problème vaut la peine d'être revu dans REBOL 3.0.

Il y a deux principes de REBOL qui sont opposés et à l'oeuvre :

1. Simplicité : nous voulons minimiser les effets de bords dans le code normal. Bon pour les débutants et bon probablement aussi pour les experts.

2. Optimisation : minimiser le coût mémoire et optimiser la performance. Si nous avons à copier des chaînes et des blocs à de nombreux endroits dans notre code, cela consomme du temps et de la mémoire.

Ainsi, il y a quelques problèmes auquel vous devriez réfléchir si vous voulez programmer finement en REBOL. Nous avons besoin d'une solution bien définie. Elle peut être n'importe laquelle de :

1. Ne rien faire. Laisser cela en l'état.

2. Modifier là où c'est le plus approprié et le plus optimal.

3. Modifier partout et ignorer le surcoût en mémoire et la diminution de performance.

Voici donc quelque chose à considérer la prochaine fois que vous vous relaxerez dans un bar, fumerez votre pipe dans votre fauteuil, ou escaladerez les montagnes dans votre coin.

Et, je vous invite à partager vos réflexions avec tout le monde, soit sur le blog dans la section "comments", ou via votre propre blog ou page web.

(Traduction : Philippe Le Goff)