The REBOL Documentation Project

-- FR - Documentation REBOL - Articles Techniques --

Articles Techniques

Créer un exécutable qui accepte des paramètres

Christophe Genser

Première publication : 7 juin 2001, et mis en ligne le vendredi 3 novembre 2006

Résumé :

Christophe Genser présente différents aspects de la conception d'un script Rebol/Command utilisant des paramètres transmis par la ligne de commande. Il étudie particulièrement les problèmes posés par l'encapsulation d'un script avec Rebol/Runtime.

1 Historique

Date	Version	Commentaires	Auteur	Email
7/06/2001	1.0.0	Article initial	Christophe Genser	
31/10/2006	1.1.0	Conversion au format MakeDoc2	Karim El Founas	karim.elfounas@easybraine.com

2 Comment relayer des paramètres à un script ?

Prenons les choses dans l'ordre et commençons par le début.

Il se trouve que Rebol peut lancer un script avec des paramètres (taper usage en commandes pour en savoir un peu plus). Ainsi, on peut exécuter un script avec la commande suivante tapée dans le Shell :

```
rebolcmd.exe -s script.r param1 param2 param3
```

Le -s demande à Rebol de ne plus tenir aucun compte de la sécurité. Dans notre cas, cela évitera à l'utilisateur néophyte d'avoir à répondre à des questions (en anglais) dont il ne connaît pas la réponse. Reste à récupérer les autres paramètres pour pouvoir les traiter. Ces paramètres se logent dans system/options/args, sous forme de chaînes de caractères.

```
>> Arguments : copy system/options/arg
== [ "param1" "param2" "param3" ]
```

Note : si on appelle rebolcmd sans préciser de script, Rebol nous annonce que param1 ne peut pas être évalué. Si on veut vraiment introduire un param1, la solution que j'ai employé consiste à commencer par un param0 qui peut être n'importe quoi et dont on ne tient pas compte, pas plus qu'on ne tient compte du l'indication de sa non évaluation.

3 Comment utiliser ces paramètres ?

L'idée (mais est-ce la seule et surtout est-ce la meilleure ?) a été d'analyser par un dialecte le bloc des paramètres avec la commande parse. Un dialecte ne peut réfléchir que sur des types word !. Il faut donc avant tout transformer notre bloc de chaînes de caractères en bloc de mots :

```
>> param: []
== []
>> foreach arg arguments [ append param to-word arg ]
== [ param1 param2 param3 ]
```

Il faut ensuite, dans le script, définir un dialecte et analyser le bloc de paramètres. Cela donne donc le scrip suivant :

```
Rebol []
; Récupération des arguments
  Param: []
  Arguments: copy system/options/arg
  foreach arg arguments [ append param to-word arg ]
; Dialecte
  syntaxe: [
     any [ ]
  ]
; Application du dialecte
  parse param syntaxe
```

4 Exemple de définition d'une syntaxe

Imaginons que nous ayons besoin de donner comme arguments à notre exécutable un numéro sous la forme d'un entier compris entre 1 et 5 et un nom de fichier qui n'est rien d'autre qu'une chaîne de caractères. Pour localiser le numéro, nous le ferons précédé du mot num. Le nom de fichier sera spécifié par le mot fic. Le script se lancera ainsi en tapant dans le Shell :

```
rebolcmd.exe -s script.r n 3 fic monfichier
```

Ou, ce qui revient au même :

```
rebolcmd.exe -s script.r fic monfichier n 3
```

Le dialecte doit donc détecter les mots fic et n, et mémoriser les mots qui les suivent. Voici le script qui correspond :

```
Syntaxe [
     Any [
          'fic set argf word! ( ) |
          'n set argn word! ( )
     ]
]
```

La règle sera fausse si l'un des mots fic ou n est rencontré sans être suivi d'un autre, ou si elle présente un mot non reconnu qui n'est pas précédé de fic ou n.

Reste à faire faire au script ce que l'on désire qu'il fasse avec ces paramètres : affecter à la variable num, de type integer !, la valeur qui suit n à condition qu'elle soit comprise entre 1 et 5, et affecter à la

variable fichier, de type file !, le nom correspondant au mot qui suit fic. Remarquons au passage qu'un mot ne peut pas être converti en entier directement. On peut biaiser en le passant en chaîne de caractères auparavant.

A l'occasion, une gestion d'erreur permet de vérifier si n est suivi d'autre chose qu'un d'un entier. Cela donne :

```
Syntaxe [
     Any [
          'fic set argf word! ( fichier: to-file argf ) |
          'n set argn word! (
               if error? try [
                    num: to-integer to-string argn
                    if any [ ( num  5 ) ]
                         [ traitement si erreur de fourchette ]
               ] [
                         traitement si erreur de type
               ]
          )
     ]
]
```

5 Peaufinage

En dernier ajustement, nous pouvons vérifier si la règle définie est correctement suivie avant de passer à autre chose. Le script final donnerait ceci :

```
Rebol []
; Récupération des arguments
Param: []
Arguments: copy system/options/arg
foreach arg arguments [ append param to-word arg ]
; Dialecte
syntaxe: [
     any [
          'fic set argf word! ( fichier: to-file argf ) |
          'n set argn word! (
             if error? try [
                num: to-integer to-string argn
                if any [ ( num  5 ) ]
                    [ traitement si erreur de fourchette ]
                ] [
                    traitement si erreur de type
                ]
             )
     ]
]
; Application du dialecte
```

```
either ( parse param syntaxe ) [
     traitement normal
] [
     traitement si erreur de syntaxe
]
```

6 Encapsulation avec Runtime

La méthode la plus pratique (à mon avis), consiste à copier le script dans le répertoire où se trouve Runtime (cmdencap.exe), et lancer celui-ci. A sa demande, taper le nom du script Rebol (script.r), puis le nom souhaité pour l'exécutable (script.exe). Une fois celui-ci placé sur une autre station, dépourvue de Rebol, un fichier network.txt doit être placé dans le même répertoire que script.exe. Un fichier vide convient. Le document setup.html livré avec Runtime fournit quelques explications à ce sujet.

On peut maintenant lancer ce programme à partir du Shell ou d'un batch en tapant :

```
script.exe -s arg n 3 fic monfichier
```

Comme lorsqu'on lance Rebol Command sans préciser de script, le premier paramètre relayé n'est pas évalué (ne me demandez pas pourquoi). Le mot arg (ou n'importe quel autre) est donc mis ici pour que ce soit lui qui ne soit pas évalué.

Et maintenant, il n'y a plus qu'à diffuser.