The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 8 - Les Séries : chaînes

Philippe Le Goff

Première publication : 12 septembre 2005, et
mis en ligne le lundi 12 septembre 2005

Résumé :

Ce document est la traduction française du Chapitre 8 du User Guide de REBOL/Core, qui concerne les séries sous forme de chaînes de caractères.

Ce document est la traduction française du Chapitre 8 du User Guide de REBOL/Core, qui concerne les séries sous forme de chaînes de caractères.
Traducteur : Philippe Le Goff

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
26 mai 2005 22:17	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Fonctions relatives aux chaînes de caractères

De nombreuses fonctions permettent de manipuler ou de créer des chaînes de caractères. Ces fonctions peuvent transformer des chaînes, y effectuer des recherches, les compresser ou les décompresser, modifier leur espacement, les analyser, et les convertir. Ces fonctions agissent sur tous les types de données liés aux chaînes de caractères, comme **string !**, **binary !**, **tag !**, **file !**, **URL !**, **email !**, et **issue !**.

Les fonctions de création, de modification et de recherche ont été présentées dans le chapitre sur les Séries. Cette présentation incluait les fonctions de chaînes :

copy : *permet la copie d'une partie ou de toute une chaîne*

make : *alloue un espace mémoire pour la chaîne, crée un type*

insert : *insére un caractère ou une sous-chaîne dans une autre*

remove : *ôte un ou plusieurs caractères d'une chaîne*

change : *change un ou plusieurs caractères dans une chaîne*

append : *insère un caractère ou une sous-chaîne à la fin d'une chaîne*

find : *trouve ou effectue une correspondance entre chaînes*

replace : *trouve une chaîne et la remplace par une autre*

De surcroît, les fonctions permettant de parcourir les séries, comme **next**, **back**, **head**, et **tail** ont déjà été présentées. Elles sont utilisées pour se déplacer dans les chaînes. Egalement, les fonctions de test des séries vous permettent de déterminer votre position dans une chaîne (Ndt : comme **index ?**, **tail ?**, **head ?**).

Ce chapitre va présenter d'autres fonctions qui permettent la conversion de valeurs REBOL en chaînes de caractères. Ces fonctions sont fréquemment utilisées, comme avec les fonctions **print** et **probe**.

Elles comprennent :

form : *convertit des valeurs avec des espaces en un format humainement lisible*

mold : *convertit des valeurs en un format REBOL*

join : *concatène des valeurs*

reform : *réduit (évalue) des valeurs avant de les traduire avec form*

remold : *réduit (évalue) des valeurs avant de les traduire avec mold*

rejoin : *réduit (évalue) des valeurs avant de les concaténer*

Ce chapitre décrira aussi les fonctions de chaîne de caractères :

detab : *remplace les tabulations par des espaces*

entab : *remplace les espaces par des tabulations*

trim : *enléve les espaces blancs ou les lignes vierges autour de chaînes de caractère*

uppercase : *change en majuscules*

lowercase : *change en minuscules*

checksum : *calcule la valeur de checksum pour une chaîne*

compress : *compresse la chaîne*

decompress : *décompresse la chaîne*

enbase : *encode une chaîne sur une autre base numérique*

debase : *convertit une chaîne encodée*

dehex : *convertit des valeurs hexadécimales ASCII en caractères (ex. dans les URLs).*

3 Conversion de valeurs en chaînes de caractères

3.1 Join

La fonction **join** prend deux arguments et les concaténe en une seule chaîne.

Le type de données de la série retournée est basé sur celui du premier argument. Quand le premier argument est une valeur de type "série", le même type est retourné :

```
str: "abc"
file: %file
url: http://www.rebol.com/

probe join str [1 2 3]
abc123
probe join file ".txt"
%file.txt
probe join url %index.html
http://www.rebol.com/index.html
```

Quand le premier argument n'est pas une série, la fonction **join** le convertit en chaîne de caractère, puis effectue la concaténation :

```
print join $11 " dollars"
$11.00 dollars
print join 9:11:01 " elapsed"
9:11:01 elapsed
print join now/date " -- today"
30-Jun-2000 -- today
print join 255.255.255.0 " netmask"
255.255.255.0 netmask
print join 412.452 " light-years away"
412.452 light-years away
```

Quand le deuxième argument ajouté est un bloc, les valeurs de ce bloc sont évaluées et ajoutées au résultat :

```
print join "a" ["b" "c" 1 2]
abc12
print join %/ [%dir1/ %sub-dir/ %filename ".txt"]
%/dir1/sub-dir/filename.txt
print join 11:09:11 ["AM" " on " now/date]
11:09:11AM on 30-Jun-2000
print join 312.423 [123 987 234]
312.423123987234
```

3.2 Rejoin

La fonction **rejoin** est identique à **join**, mis à part le fait qu'elle prend un argument de type bloc, qui est évalué :

```
print rejoin ["try" 1 2 3]
try123
print rejoin ["h" 'e #"l" (to-char 108) "o"]
hello
```

3.3 Form

La fonction **form** transforme une valeur en chaîne de caractères :

```
print form $1.50
$1.50
print type? $1.50
money
print type? form $1.50
string
```

L'exemple suivant utilise **form** pour trouver un nombre par sa valeur décimale :

```
blk: [11.22 44.11 11.33 11.11]
foreach num blk [if find form num ".11" [print num]]
44.11
11.11
```

Lorsque **form** est utilisée sur un bloc, toutes les valeurs du bloc sont converties en chaînes de caractères, avec des espaces entre chacune d'elles :

```
print form [11.22 44.11 11.33]
11.22 44.11 11.33
```

La fonction **form** n'évalue pas les valeurs d'un bloc. Elle transforme des mots en chaînes de caractères :

```
print form [a block of undefined words]
a block of undefined words
print form [33.44 num "-- unevaluated string:" str]
33.44 num -- unevaluated string: str
```

3.4 Reform

La fonction **reform** est identique à **form**, excepté que les blocs sont réduits (évalués) avant d'être convertis en chaînes.

```
str1: "Today's date is:"
str2: "The time is now:"
print reform [str1 now/date newline str2 now/time]
Today's date is: 30-Jun-2000 The time is now: 14:41:44
```

La fonction d'affichage **print** est basée sur la fonction **reform**.

3.5 Mold

La fonction **mold** convertit une valeur en chaîne de caractère utilisable par REBOL. Les chaînes créées avec **mold** peuvent être retransformées en valeurs REBOL avec la fonction **load**.

```
blk: [[11 * 4] ($15 - $3.89) "eleven dollars"]
probe blk
[[11 * 4] ($15.00 - $3.89) "eleven dollars"]
molded-blk: mold blk
probe molded-blk
{[[11 * 4] ($15.00 - $3.89) "eleven dollars"]}
print type? blk
block
print type? molded-blk
string
probe first blk
[11 * 4]
probe first molded-blk
#"["
```

Les chaînes renvoyées par **mold** peuvent être récupérées par REBOL :

```
new-blk: load molded-blk
probe new-blk
[[11 * 4] ($15.00 - $3.89) "eleven dollars"]
print type? new-blk
block
probe first new-blk
[11 * 4]
```

La fonction **mold** n'évalue pas les valeurs d'un bloc :

```
money: $11.11
sub-blk: [inside another block mold this is unevaluated]
probe mold [$22.22 money "-- unevaluated block:" sub-blk]
{[$22.22 money "-- unevaluated block:" sub-blk]}
probe mold [a block of undefined words]
[a block of undefined words]
```

3.6 Remold

La fonction **remold** s'utilise comme **mold**, à l'exception du fait que les blocs sont réduits (évalués) avant d'être convertis.

```
str1: "Today's date is:"
probe remold [str1 now/date]
{["Today's date is:" 30-Jun-2000]}
```

3.7 Fonctions pour gérer les espaces

3.7.1 Trim

La fonction **trim** enlève tous les espaces multiples d'une chaîne. Par défaut, **trim** enlève les espaces en excés au début et à la fin d'une chaîne :

```
str: "  line of text with spaces around it "
print trim str
line of text with spaces around it
```

Remarquez que la chaîne est modifiée dans le processus :

```
print str
line of text with spaces around it
```

Pour exécuter **trim** sur une copie d'une chaîne, écrivez :

```
print trim copy str
line of text with spaces around it
```

La fonction **trim** inclut quelques raffinement afin d'indiquer où les espaces doivent être supprimés dans la chaîne :

/head : *enlève les espaces en début de chaîne*

/tail : *enlève les espaces en fin de chaîne*

/auto : *enlève les espaces à chaque ligne, relativement à la première ligne*

/lines : *enlève les sauts de lignes, et les remplace par des espaces*

/all : *enlève tous les espaces, les tabulations,et les sauts de lignes*

/with : *enlève tous les caractères spécifiés*

Utilisez les raffinements **/head** et **/tail** pour ôter les espaces en début et en fin de chaîne :

```
probe trim/head copy str
line of text with spaces around it
probe trim/tail copy str
line of text with spaces around it
```

Utilisez le raffinement **/auto** pour effacer les espaces résiduels sur des lignes multiples, tout en conservant intacte l'indentation :

```
str: {
    indent text
        indent text
            indent text
        indent text
    indent text
}
print str
indent text
    indent text
        indent text
    indent text
indent text
probe trim/auto copy str
{indent text
    indent text
        indent text
    indent text
indent text
}
```

Le raffinement **/lines** permet d'enlever les espaces en début et en fin de chaîne, mais également de convertir les sauts de lignes en espaces :

```
probe trim/lines copy str
{indent text indent text indent text indent text indent text}
```

L'usage du raffinement **/all** enléve tous les espaces, les tabulations, les sauts de lignes :

```
probe trim/all copy str
indenttextindenttextindenttextindenttextindenttext
```

La raffinement **/with** permet d'éliminer de la chaîne tous les caractères qui ont été spécifiés avec **/with**. Dans l'exemple suivant, les espaces, les sauts de lignes, et les caractères "e" et "t" sont supprimés :

```
probe trim/with copy str " ^/et"
indnxindnxindnxindnxindnx
```

3.7.2 Detab et Entab

Les fonctions **detab** et **entab** transforment les tabulations en espaces et inversement, les espaces en tabulations.

```
str:
{^(tab)line one
^(tab)^(tab)line two
^(tab)^(tab)^(tab)line three
^(tab)line^(tab)full^(tab)of^(tab)tabs}
print str
line one
        line two
            line three
    line    full    of  tabs
```

Par défaut, la fonction **detab** convertit chaque tabulation en une série de **quatre** espaces (le standard pour le style REBOL). Toutes les tabulations dans la chaîne seront transformées en espaces, quelque soient leurs positions.

```
probe detab str
{    line one
        line two
            line three
    line    full    of  tabs}
```

Remarquez que les fonctions **detab** et **entab** modifient la chaîne qu'elles prennent en argument. Pour travailler sur une copie de la chaîne source, utilisez la fonction **copy**.

La fonction **entab** convertit les espaces en tabulations. Chaque série de quatre espaces sera transformée en une tabulation. Seuls les espaces en début de ligne seront transformés en tabulations.

```
probe entab str
{^-line one
^-^-line two
^-^-^-line three
^-line^-full^-of^-tabs}
```

Vous pouvez utiliser le raffinement **/size** pour spécifier la taille des tabulations. Par exemple, si vous voulez convertir chaque tabulation en série de huit espaces, ou transformer ces huit espaces en une tabulation, vous pouvez utiliser cet exemple :

```
probe detab/size str 8
{        line one
                line two
                        line three
        line    full    of      tabs}
probe entab/size str 8
{^-line one
^-^-line two
^-^-^-line three
^-line^-full^-of^-tabs}
```

3.8 Uppercase et Lowercase

Deux fonctions permettent le changement de la casse des caractères : **uppercase** et **lowercase**.

La fonction **uppercase** prend une chaîne en argument et la met en majuscule.

```
print uppercase "SamPle TExT, tO test CASES"
SAMPLE TEXT, TO TEST CASES
```

La fonction **lowercase** effectue le travail inverse, elle met les caractères en minuscule :

```
print lowercase "Sample TEXT, tO teST Cases"
sample text, to test cases
```

Pour transformer juste une partie de la chaîne, utilisez le raffinement **/part** :

```
print upppercase/part "ukiah" 1
Ukiah
```

3.9 Checksum

La fonction **checksum** renvoie la valeur de checksum (somme de contrôle) d'une chaîne. Plusieurs types de checksum peuvent être calculés

CRC24 : *contrôle de redondance cyclique (défaut)*

TCP : *checksum Internet TCP 16-bit.*

Secure : *Retourne un checksum sécurisé par cryptage*

NdT :

Les élèments qui suivent correspondent à l'aide en ligne sur la fonction **checksum**, et non à la documentation officielle du User Guide, qui semble caduque sur ce point.

Checksum permet l'usage des raffinements :

/tcp : *renvoie la valeur de checksum Internet 16 bits*

/secure : *renvoie la valeur de checksum sécurisé par cryptage*

/hash : *retourne une valeur d'index pour une table de hachage.*

/method : *utilise une méthode de cryptage, qui peut être SHA1 ou MD5*

/key : *utilise une clé pour une authentification HMAC (Keyed-Hashing for Message Authentication Code).*

NdT :

Voir l'aide en ligne pour plus de détails.

Par défaut, c'est la valeur de checksum CRC qui est calculée :

```
print checksum "hello"
52719
print checksum (read http://www.rebol.com/)
356358
```

Pour calculer la checksum TCP, utilisez le raffinement **/tcp** :

```
print checksum/tcp "hello"
10943
```

Le raffinement **/secure** retournera une valeur binaire, pas un entier. Utilisez le raffinement **/secure**, pour calculer ce type de checksum :

```
print checksum/secure "hello"
#{AAF4C61DDCC5E8A2DABEDE0F3B482CD9AEA9434D}
```

Le raffinement **/method** permet d'utiliser (par exemple pour contrôler des mots de passe) une méthode de cryptage MD5 ou SHA1 :

```
print checksum/method "password2005" 'md5
#{9FDC0F6F1A5A0443F1C6E2393BE936DE}
print checksum/method "password2005" 'sha1
#{3E352A5705741521337C01537AEA0A54DD13B993}
```

3.10 Compression et décompression

La fonction **compress** va compresser une chaîne et retourner un type de donnée binaire. Dans l'exemple suivant, un petit fichier est compressé :

```
Str:
{I wanted the gold, and I sought it,
  I scrabbled and mucked like a slave.
Was it famine or scurvy -- I fought it;
  I hurled my youth into a grave.
I wanted the gold, and I got it --
  Came out with a fortune last fall, --
Yet somehow life's not what I thought it,
  And somehow the gold isn't all.}

print [size? str "bytes"]
306 bytes
bin: compress str

print [size? bin "bytes"]
156 bytes
```

Remarquez que le résultat de la compression est du type de données binaire (**binary !**).

La fonction **decompress** décompresse une chaîne qui a été préalablement compressée.

```
print decompress bin
I wanted the gold, and I sought it,
  I scrabbled and mucked like a slave.
Was it famine or scurvy -- I fought it;
  I hurled my youth into a grave.
I wanted the gold, and I got it --
  Came out with a fortune last fall, --
Yet somehow life's not what I thought it,
  And somehow the gold isn't all.
```

Sauvegarde de vos données :

Conservez toujours une copie non compressée de vos données compressées. Si vous perdez un seul octet d'une chaîne binaire compressée, il sera difficile de récupérer les données. Ne sauvegardez pas des fichiers d'archives en format compressé à moins que vous n'ayiez des copies d'origine non compressées.

3.11 Modification de la base numérique

Pour être envoyé sous forme de texte, les chaînes binaires doivent être encodées en hexadécimal ou en base 64. Ceci est fréquemment réalisé pour le courrier électronique, ou le contenu de groupes de nouvelles (newsgroups).

La fonction **enbase** encodera une chaîne binaire :

```
line: "No! There's a land!"
print enbase line
Tm8hIFRoZXJlJ3MgYSBsYW5kIQ==
```

Les chaînes encodées peuvent être décodées avec la fonction **debase**. Notez que le résultat est une valeur de type binaire. Pour convertir cette valeur en une chaîne de caractéres (**string !**), utilisez la fonction **to-string**.

```
b-line: debase e-line
print type? b-line
binary
probe b-line
#{4E6F2120546865726527732061206C616E6421}
print to-string b-line
No! There's a land!
```

Le raffinement **/base** peut être utiisé avec les fonctions **enbase** et **debase**, pour spécifier un encode en base-2 (binaire), base-16 (hexadécimal) ou base-64.

Voici quelques exemples utilisant l'encodage en base 2 :

```
e2-str: enbase/base str 2
print e2-str
01100001
b2-str: debase/base e2-str 2
print type? b2-str
binary
probe b2-str
#{61}
print to-string b2-str
a
```

Quelques exemples avec un encodage en hexadécimal (base-16) :

```
e16-line: enbase/base line 16
print e16-line
4E6F2120546865726527732061206C616E6421
b16-line: debase/base e16-line 16
print type? b16-line
binary
probe b16-line
#{4E6F2120546865726527732061206C616E6421}
```

```
print to-string b16-line
No! There's a land!
```

3.12 Décodage hexadécimal pour Internet

La fonction **dehex** convertit les caractères encodés en hexadécimal des URLs Internet ou CGI en chaînes de caractères. La représentation hexadécimal ASCII se présente dans un URL ou une chaîne CGI comme %xx, où xx est une valeur hexadécimale.

```
str: "there%20seem%20to%20be%20no%20spaces"
print dehex str
there seem to be no spaces
print dehex "%68%65%6C%6C%6F"
hello
```