The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 3 - Présentation rapide

Philippe Le Goff

Première publication : 14 septembre 2005, et mis en ligne le mercredi 14 septembre 2005

Résumé :

Ce document est la traduction française du Chapitre 3 du User Guide de REBOL/Core, la présentation rapide du langage.

Ce document est la traduction française du Chapitre 3 du User Guide de REBOL/Core, la présentation rapide du langage.
Traducteur : Philippe Le Goff

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email

18 mai 2005 - 19:31	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Présentation générale

Ce chapitre a pour objectif de vous familiariser rapidement avec le langage REBOL. A l'aide d'exemples, ce chapitre présente les concepts de base et la structure du langage, et ceci, depuis le concept concernant les valeurs des données jusqu'à la maîtrise des opérations liées au réseau.

3 Valeurs

Un script est écrit avec un ensemble de valeurs. De nombreux types de valeurs existent et vous êtes confrontés à beaucoup d'entre eux dans le quotidien de vos journées. Lorsque cela est possible, REBOL permet aussi d'utiliser des formats internationaux pour des valeurs telles que les nombres décimaux, les données monétaires, les dates et les heures.

3.1 Nombres

Les nombres sont écrits sous forme d'entiers, de décimaux ou en notation scientifique. Par exemple :

```
1234 -432 3.1415 1.23E12
```

Et vous pouvez aussi écrire au format Européen :

```
123,4 0,01 1,2E12
```

3.2 Heures

Les données horaires sont écrites en heures et en minutes, avec les secondes en option, chacunes étant séparées par le symbole " **:**".

Par exemple :

```
12:34 20:05:32 0:25.345 0:25,345
```

Les secondes peuvent inclure les fractions de secondes (milliseconde). Les heures peuvent aussi être indiquées au format anglo-saxon AM et PM (sans espace) :

```
12:35PM 9:15AM
```

3.3 Dates

Les dates sont écrites dans l'un ou l'autre des formats internationaux : *jour-mois-année* ou *année-mois-jour*. Une date peut aussi inclure une heure et une indication de fuseau horaire. Le nom ou l'abréviation du mois peuvent être utilisés de façon à rendre le format plus semblable à celui en usage aux Etats-Unis. Par exemple :

```
20-Apr-1998 20/Apr/1998 (USA)

20-4-1998 1998-4-20     (international)

1980-4-20/12:32         (date et heure)

1998-3-20/8:32-8:00     (avec fuseau horaire)
```

3.4 Données monétaires

Les données monétaires sont écrites avec un symbole optionnel de trois lettres représentant la devise, suivies d'une grandeur numérique. Par exemple :

```
$12.34  USD$12.34  CAD$123.45  DEM$1234,56
```

3.5 Tuples

Les "tuples" sont utilisés pour des numéros de version, des valeurs RGB (Rouge-Vert-Bleu) de couleur, et des adresses réseau. Ils sont écrits sous la forme de nombres entiers compris entre 0 et 255 et séparés par des points.

Par exemple :

```
2.3.0.3.1  255.255.0  199.4.80.7
```

Au moins deux points sont requis (sinon, le nombre sera interprété comme une valeur décimale, pas un tuple). Exemple :

```
2.3.0  ; tuple
2.3.   ; tuple
2.3    ; decimal
```

3.6 Chaînes de caractères

Les chaînes de caractères sont écrites en une seule ou plusieurs lignes. Les chaînes monolignes sont incluses entre des guillemets ("). Les chaînes s'étendant sur plusieurs lignes sont incluses entre des accolades .

Les chaînes qui comprennent des apostrophes, des marques de tabulations ou des sauts de lignes doivent être incluses entre des accolades comme pour le format multi-lignes. Par exemple :

```
"Here is a single-line string"

{Here is a multiline string that
contains a "quoted" string.}
```

Les caractères spéciaux (échappements) à l'intérieur de chaînes sont spécifiés avec le caractère (^). Voir le paragraphe relatif aux chaînes de caractères dans le chapitre sur les Valeurs, pour la table des caractères spéciaux.

3.7 Balises

Les balises sont utilisées dans les langages comme XML et HTML. Les balises sont incluses entre les caratères "". Par exemple :

3.8 Adresses email

Les adresses email sont écrites directement en REBOL. Elles doivent inclure le symbole "@".

Par exemple :

```
info@rebol.com

pres-bill@oval.whitehouse.gov
```

3.9 URLs

La plupart des types d'URLs Internet est accepté directement sous REBOL. Celles-ci doivent commencer par un nom de protocole (HTTP par exemple) suivi d'un chemin (path).

Par exemple :

```
http://www.rebol.com

ftp://ftp.rebol.com/sendmail.r

ftp://freda:grid@da.site.dom/dir/files/
```

```
mailto:info@rebol.com
```

3.10 Noms de fichiers

Les noms de fichiers sont précédés par le symbole "%" (pourcentage), qui permet de faire la distinction avec les autres mots. Par exemple :

```
%data.txt

%images/photo.jpg

%../scripts/*.r
```

3.11 Paires

Les paires sont utilisées pour indiquer des coordonnées spatiales, comme des positions sur un écran. Elle sont utilisées pour indiquer aussi bien des positions ou des tailles. Les coordonnées sont séparées par un "**x**".

Par exemple :

```
100x50

1024x800

-50x200
```

3.12 Issues

Les "issues" sont des numéros d'identification, des séquences particulières de caractères, comme des numéros de téléphone, des numéros de séries, ou de carte de crédit.

Par exemple :

```
#707-467-8000

#0000-1234-5678-9999

#MFG-932-741-A
```

3.13 Binaires

Les binaires sont des chaînes d'octets de n'importe quelle longueur. Ils peuvent être encodés directement en hexadécimal ou en base-64.

Par exemple :

```
#{42652061205245424F4C}

64#{UkVCT0wgUm9ja3Mh}
```

4 Mots

Les mots sont les symboles qu'utilise REBOL. Un mot peut être ou non une variable, selon l'usage qui en est fait. Les mots peuvent aussi être directement utilisés comme symboles.

```
show next image

Install all files here

Country State City Street Zipcode

on off true false one none
```

REBOL n'a pas de mots-clés ; il n'existe pas de restriction sur les mots utilisés, et sur la façon dont ils sont utilisés. Par exemple, vous pouvez définir votre propre fonction, appelée **print**, et l'utiliser au lieu de celle prédéfinie dans le langage pour l'affichage de valeurs.

Les mots sont insensibles à la casse, et peuvent inclure des traits d'union et d'autres caractères spéciaux, comme :

```
+ - `  * ! ~ &amp; ? |
```

Les exemples suivants présentent quelques mots valides :

```
number?  time?  date!
image-files  l'image
++ -- == +-
***** *new-line*
left&amp;right left|right
```

La fin d'un mot est indiqué par un espace, un saut de ligne, ou l'un des caractères suivants :

```
[ ] ( ) { } " : ; /
```

Les caractères suivants, par contre, ne sont pas autorisés dans les mots :

```
@ # $ % ^ ,
```

5 Blocs

REBOL se compose de groupes de valeurs et de mots placés dans des blocs. Les blocs sont utilisés pour le code, les listes, les matrices, les tableaux, les répertoires, les associations et autres formes ordonnées. Un bloc est une sorte de série, dans laquelle les valeurs sont organisées dans un ordre spécifique.

Un bloc est inclus entre deux crochets [ ]. A l'intérieur du bloc, les valeurs et les mots peuvent être organisés selon n'importe quel ordre. Les exemples suivants illustrent différentes formes de blocs valides :

```
[white red green blue yellow orange black]

["Spielberg" "Back to the Future" 1:56:20 MCA]

[
    Ted    ted@gw2.dom   #213-555-1010
    Bill   billg@ms.dom  #315-555-1234
    Steve  jobs@apl.dom  #408-555-4321
]

[
    "Elton John"  6894  0:55:68
    "Celine Dion" 68861 0:61:35
    "Pink Floyd"  46001 0:50:12
]
```

Les blocs sont utilisés pour du code aussi bien que pour des données, comme le montrent les exemples suivants :

```
loop 10 [print "hello"]

if time > 10:30 [send jim news]

sites: [
    http://www.rebol.com [save %reb.html data]
    http://www.cnn.com   [print data]
    ftp://www.amiga.com  [send cs@org.foo data]
]

foreach [site action] sites [
    data: read site
    do action
]
```

Un fichier script est aussi un bloc.

Bien qu'il n'inclut pas de crochets, le bloc est implicite. Par exemple, si les lignes ci-dessous sont mises dans un fichier script :

```
red
green
blue
yellow
```

Lorsque le fichier sera évalué, ce sera sous la forme d'un bloc contenant les mots *red*, *green*, *blue*, et *yellow*. Cela revient à écrire :

```
[red green blue yellow]
```

6 Variables

Les mots peuvent être utilisés comme variables faisant référence à des valeurs. Pour définir un mot en tant que variable, faites-le suivre de deux points (:), puis de la valeur attribuée à la variable, comme indiqué dans les exemples suivants :

```
age: 22
snack-time: 12:32
birthday: 20-Mar-1997
friends: ["John" "Paula" "Georgia"]
```

Une variable peut faire référence à n'importe quel type de valeur, notamment des fonctions (voir chapitre sur les Fonctions) ou des objets (voir celui sur les Objets).

Une variable fait référence à une valeur, spécifique à un contexte donné qui peut être un bloc, une fonction, ou un programme complet.

En dehors de ce contexte, la variable peut faire référence à d'autres valeurs, ou à aucune. Le contexte d'une variable peut s'étendre sur le programme entier (portée globale) ou être restreint à un bloc particulier, une fonction, ou un objet. Dans d'autres langages, le contexte d'une variable est souvent identifié comme "*la portée d'une variable*".

7 Evaluation

Les blocs sont évalués pour renvoyer leurs résultats. Quand un bloc est évalué, les valeurs de ses variables sont obtenues. Les exemples suivants évaluent les variables *age*, *snack-time*, *birthday*, et *friends* qui ont été définies dans le paragraphe précédent :

```
print age
22
if current-time > snack-time [print snack-time]
12:32
print third friends
Georgia
```

Un bloc peut être évalué plusieurs fois en utilisant une boucle, comme le montrent les exemples

suivants :

```
loop 10 [prin "*"]  ;("prin" n'est pas une erreur de frappe, voir le manuel)
**********
loop 20 [
    wait 8:00
    send friend@rebol.com read http://www.cnn.com
]

repeat count 3 [print ["count:" count]]
count: 1
count: 2
count: 3
```

L'évaluation d'un bloc renvoie un résultat. Dans les exemples suivants, 5 et PM sont les résultats de l'évaluation, respectivement, de chaque bloc :

```
print do [2 + 3]
5
print either now/time  PM
```

En REBOL, il n'y a pas d'opérateur particulier pour les régles de priorité, dans l'évaluation d'un bloc. Les valeurs et les mots d'un bloc sont toujours évalués du premier au dernier, comme l'illustre l'exemple :

```
print 2 + 3 * 10
50
```

L'usage de parenthèses peut permettre de contrôler l'ordre de l'évaluation, comme dans les exemples ci-dessous :

```
2 + (3 * 10)
32
(length? "boat") + 2
6
```

Vous pouvez aussi évaluer un bloc et retourner chacun des résultats calculés à l'intérieur du bloc. C'est l'objet de la fonction **reduce** :

```
reduce [1 + 2  3 + 4  5 + 6]
3 7 11
```

8 Fonctions

Une fonction est un bloc avec des variables qui donnent de nouvelles valeurs à chaque fois que le bloc est évalué.

Ces variables sont appelées les *arguments* de la fonction. Dans l'exemple suivant, le mot *sum* fait

référence à une fonction qui va accepter deux arguments, *a* et *b* :

```
sum: func [a b] [a + b]
```

Dans l'exemple précédent, **func** est utilisée pour définir une nouvelle fonction.

Le premier bloc dans la fonction définit les arguments de cette fonction. Le second bloc est le bloc de code qui sera évalué quand la fonction sera utilisée. Dans cet exemple, le second bloc additionne les deux valeurs en arguments et renvoie le résultat.

Le prochain exemple illustre le fonctionnement de la fonction *sum* qui a été définie auparavant :

```
print sum 2 3
5
```

Certaines fonctions nécessitent des variables locales tout comme des arguments. Pour définir ce genre de fonction, vous pouvez utiliser **function** au lieu de **func** (NdT : dépassé ! Utilisez plutôt **func** avec le **refinement** /local), comme le montre l'exemple suivant :

```
average: function [series] [total] [
    total: 0
    foreach value series [total: total + value]
    total / (length? series)
]

print average [37 1 42 108]
47
```

Dans l'exemple ci-dessus, le mot *series* est un argument et le mot *total* est une variable locale utilisée par la fonction, à des fins de calcul.

Le bloc d'argument de la fonction peut contenir des libellés décrivant l'objet et l'usage de la fonction, et de ses arguments, comme illustré ci-dessous :

```
average: function [
    "Return the numerical average of numbers"
    series "Numbers to average"
] [total] [
    total: 0
    foreach value series [total: total + value]
    total / (length? series)
]
```

Les chaînes de caractères sont stockées avec la fonction et peuvent être consultées lors d'une demande d'aide, au sujet de la fonction, avec l'aide en ligne :

```
help average
USAGE:
    AVERAGE series
DESCRIPTION:
```

```
    Return the numerical average of numbers
    AVERAGE is a function value.
ARGUMENTS:
    series -- Numbers to average (Type: any)
```

9 Paths

NdT : le terme anglais "path" a été conservé.

Si vous utilisez des fichiers et des URLs, alors vous êtes déjà familier avec le concept de "paths" (chemins). Un path fournit un ensemble de valeurs qui sont utilisées pour "naviguer" d'un point à un autre. Dans le cas d'un fichier, un "path" spécifie le parcours au travers les différents répertoires jusqu'à l'endroit où se trouve le fichier.

En REBOL, les valeurs dans un path sont appelées des raffinements.

Le symbole slash (***/***) est utilisé pour séparer les mots et les valeurs dans un path, comme le montrent les exemples suivants pour un fichier ou une URL.

```
%source/images/globe.jpg

http://www.rebol.com/examples/simple.r
```

Les paths sont aussi utilisés pour sélectionner des valeurs de blocs, ou des caractères dans des chaînes , accéder à des variables au sein d'objets, et encore pour les raffinements d'une fonction, comme le montrent les exemples :

```
USA/CA/Ukiah/size (block selection)
names/12          (string position)
account/balance   (object function)
match/any         (function option)
```

La fonction **print** dans l'exemple ci-dessous montre la simplicité d'utilisation des paths pour accéder à une mini base de donnée sà partir de bloc :

```
towns: [
    Hopland [
    phone #555-1234
    web   http://www.hopland.ca.gov
    ]

    Ukiah [
    phone #555-4321
    web   http://www.ukiah.com
```

```
    email info@ukiah.com
    ]
]

print towns/ukiah/web
http://www.ukiah.com
```

10 Objets

Un objet est un ensemble de variables dont les valeurs sont spécifiques à un contexte. Les objets sont utilisés pour manipuler des structures de données et des comportements complexes.

L'exemple suivant montre comment un compte bancaire est décrit par un objet dont les attributs et les fonctions sont spécifiés :

```
account: make object! [
  name: "James"
  balance: $100
  ss-number: #1234-XX-4321
  deposit:  func [amount] [balance: balance + amount]
  withdraw: func [amount] [balance: balance - amount]
]
```

Dans l'exemple ci-dessus, les mots *name*, *balance*, *ss-numer*, *deposit*, et *withdraw* sont des variables locales de l'objet *account*. Les variables *deposit* et *withdraw* sont des fonctions qui sont définies au sein de l'objet. Les variables de l'objet *account* sont accessibles avec un path :

```
print account/balance
$100.00
account/deposit $300

print ["Balance for" account/name "is" account/balance]
Balance for James is $400.00
```

L'exemple suivant montre comment créer un autre compte avec un nouveau solde, mais possédant toutes les autres valeurs provenant du premier compte.

```
checking-account: make account [
    balance: $2000
]
```

Vous pouvez aussi créer un compte bancaire qui étend les caractéristiques de l'objet *account* en ajoutant le nom de la banque et la date de la dernière opération, comme illustré ci-dessous :

```
checking-account: make account [
    bank: "Savings Bank"
    last-active: 20-Jun-2000
]
```

```
print checking-account/balance
$2000.00
print checking-account/bank
Savings Bank
print checking-account/last-active
20-Jun-2000
```

11 Scripts

Un script est un fichier qui contient un bloc pouvant être chargé et évalué. Le bloc peut contenir du code ou des données, et aussi typiquement un certain nombre de sous-blocs.

Les scripts nécessitent un en-tête (**header**) pour identifier la présence de code. L'en-tête peut inclure le titre du script, la date, et d'autres informations. Dans l'exemple suivant, le premier bloc contient l'information propre à l'en-tête.

```
REBOL [
    Title: "Web Page Change Detector"
    File:  %webcheck.r
    Author: "Reburu"
    Date:  20-May-1999
    Purpose: {
    Determine if a web page has changed since it was
    last checked, and if it has, send the new page
    via email.
    }
    Category: [web email file net 2]
]

page: read http://www.rebol.com

page-sum: checksum page

if any [
    not exists? %page-sum.r
    page-sum  (load %page-sum.r)
][
    print ["Page Changed" now]
    save %page-sum.r page-sum
    send luke@rebol.com page
]
```

12 Fichiers

En REBOL, les fichiers peuvent facilement être manipulés. La liste suivante décrit quelques

opérations relatives aux fichiers.

Récupérer le contenu d'un fichier texte :

```
data: read %plan.txt
```

Vous pouvez afficher un fichier texte avec :

```
print read %plan.txt
```

Pour écrire dans un fichier texte :

```
write %plan.txt data
```

Par exemple, vous pouvez écrire l'heure courante avec :

```
write %plan.txt now
```

Vous pouvez aussi facilement ajouter des données en fin de fichier :

```
write/append %plan data
```

Les fichiers binaires peuvent être lus ou écrits avec :

```
data: read/binary %image.jpg
```

```
write/binary %new.jpg data
```

Pour charger un fichier comme un bloc ou une valeur REBOL :

```
data: load %data.r
```

Sauvegarder un bloc ou une valeur dans un fichier est tout aussi facile :

```
save %data.r data
```

Pour évaluer un fichier en tant que script (ce qui nécessite un en-tête valide pour cela) :

```
do %script.r
```

Vous pouvez lister le contenu d'un répertoire avec :

```
dir: read %images/
```

et, vous pouvez alors afficher les noms de fichiers avec :

```
foreach file dir [print file]
```

Pour créer un répertoire :

```
make-dir %newdir/
```

Pour savoir quel est le répertoire courant :

```
print what-dir
```

Si vous avez besoin d'effacer un fichier :

```
delete %oldfile.txt
```

Vous pouvez aussi renommer un fichier avec :

```
rename %old.txt %new.txt
```

Pour avoir les propriétés d'un fichier :

```
print size? %file.txt
```

```
print modified? %file.txt
```

```
print dir? %image
```

13 Réseau

Il existe plusieurs protocoles pré-inclus dans REBOL. Ces protocoles sont faciles à utiliser et nécessitent juste un minimum de connaissances relatives au Réseau.

13.1 HTTP

L'exemple suivant montre comment utiliser le protocole HTTP pour lire une page web :

```
page: read http://www.rebol.com
```

L'exemple ci-dessous récupére une image dans une page web, et la sauvegarde en local dans un fichier :

```
image: read/binary http://www.page.dom/image.jpg
```

```
write/binary %image.jpg image
```

13.2 FTP

Ici, le code suivant lit et écrit des fichiers sur un serveur en utilisant le protocole FTP :

```
file: read ftp://ftp.rebol.com/test.txt

write ftp://user:pass@site.dom/test.txt file
```

L'exemple suivant retourne le contenu d'un répertoire en FTP :

```
print read ftp://ftp.rebol.com/pub
```

13.3 SMTP

Les exemples suivants envoient un courrier électronique avec le protocole SMTP :

```
send luke@rebol.com "Use the force."
```

Ici le contenu d'un fichier texte est envoyé par email

```
send luke@rebol.com read %plan.txt
```

13.4 POP

L'exemple suivant récupére des emails avec le protocole POP et affiche tous les messages courants de la boîte de réception, en les laissant sur le serveur :

```
foreach message read pop://user:pass@mail.dom [
    print message
]
```

13.5 NNTP

L'exemple qui suit récupére les news en utilisant le protocole NNTP (network news transfer protocol), en lisant toutes les nouvelles relatives à un groupe en particulier :

```
messages: read nntp://news.server.dom/comp.lang.rebol
```

Ici, l'exemple ci-dessous permet la lecture et l'impression de tous les groupes de discussions :

```
news-groups: read nntp://news.some-isp.net

foreach group news-groups [print group]
```

13.6 DAYTIME

L'exemple suivant retourne l'heure courante d'un serveur :

```
print read daytime://everest.cclabs.missouri.edu
```

13.7 WHOIS

Il est possible avec l'exemple ci-dessous de savoir qui est responsable d'un domaine, avec le protocole Whois :

```
print read whois://rebol@rs.internic.net
```

13.8 FINGER

L'exemple suivant ramène des informations concernant un utilisateur avec le protocole Finger :

```
print read finger://username@host.dom
```

13.9 DNS

Ici, une adresse Internet est déterminée à partir d'un nom de domaine, et inversement un nom de domaine à partir d'une adresse IP :

```
print read dns://www.rebol.com

print read dns://207.69.132.8
```

13.10 TCP

Les connections directes avec TCP/IP sont également possibles en REBOL.

L'exemple qui illustre ceci est un serveur simple, mais pratique, qui attend des connections sur un port, et exécute ce qui a été recu :

```
server-port: open/lines tcp://:9999

forever [
    connection-port: first server-port
    until [
    wait connection-port
    error? try [do first connection-port]
    ]
    close connection-port
```

```
]
```