The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 14 - Les Ports

Philippe Le Goff

Première publication : 27 mai 2005, et mis en ligne le vendredi 27 mai 2005

Résumé :

Ce document est la traduction française du Chapitre 14 du User Guide de REBOL/Core, qui concerne les Ports.

Ce document est la traduction française du Chapitre 14 du User Guide de REBOL/Core, qui concerne les Ports. Traducteur : Philippe Le Goff

Historique de la traduction

Date	Version	Commentaires	Auteur	Email
29 avril 2005 17:55	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

Généralités

Les ports permettent d'accéder à des séries externes résultant de fichiers, du réseau, de consoles, de périphériques externes, d'évenements, de codecs, ou de bases de données.

Les données d'un port sont traitées en utilisant les fonctions standards de REBOL propres aux séries, comme décrit dans le chapitre sur les Séries.

Les ports sont utilisés aussi bien pour des entrées que pour des sorties de données.

Les types de données manipulés par un port dépendent de la façon dont celui-ci a été ouvert.

Trois types de données sont possibles :

Type	Description
String	une série d'octets, avec conversion des sauts de lignes (par défaut)
Binary	une série d'octets, sans modification de la donnée
Block	une série de valeurs REBOL

Un port peut être ouvert dans l'un des deux modes suivants (*buffering modes*) :

Mode	Description
Mise en buffer (buffered)	toutes les données sont conservées en mémoire (par défaut)
Direct	les données ne sont pas conservées en mémoire

De plus, un port peut être ouvert avec :

Type	Description
Wait	le port attend, écoute, pour savoir si des données arrivent (par défaut)
No-wait	aucune attente de données

Ouverture d'un port

La fonction open

La fonction **open** initialise l'accès à un port selon des paramètres, qui lui ont été spécifiés. La fonction peut être invoquée avec un nom de fichier, une URL, ou un objet. De plus, il existe plusieurs raffinements qui affecterons l'opération d'ouverture ou l'accès au contenu du port.

La méthode la plus simple pour utiliser **open** est de lui fournir un nom de fichier, ou une URL en argument.

Dans l'exemple ci-dessous, un port de type fichier est ouvert :

```
fp: open %file.txt
```

La variable *fp* référence le port. Si le port n'est pas ouvert, une erreur se produira. Si besoin, l'erreur peut être capturée avec la fonction **try**.

Par défaut, le port est ouvert en mode "*buffered*".

Cela signifie qu'un fichier est accèdé et modifié en mémoire, et que les changements ne sont pas écrits dans le fichier tant que le port n'est pas fermé ou mis à jour.

Pour les fichiers, la fonction **open** créera automatiquement le fichier si celui-ci n'existe pas auparavant.

```
close open %somefile.txt
if exists? %somefile.txt [print "somefile exists"]
somefile exists
```

Le raffinement **/new** peut être utilisé pour remplacer un fichier existant.

```
write %somefile.txt "text in some file"
print read %somefile.txt
text in some file
close insert open/new %somefile.txt "new data"
print read %somefile.txt
new data
```

Une fois le port ouvert, les opérations relatives aux séries comme **copy**, **insert**, **remove**, **clear**, **first**, **next**, et **length ?** peuvent être utilisées pour accèder au contenu du port et le modifier.

Raffinements de la fonction open

La fonction **open** accepte un certain nombre de raffinements qui peuvent être utilisés pour modifier son comportement :

Raffinement	Description
/binary	les données du port sont binaires
/string	les données sont de type texte, les fins de ligne sont automatiquement converties
/with	précise une fin de ligne spéciale
/lines	manipule les données une ligne à la fois ou comme un bloc de lignes
/direct	ne met pas en mémoire
/new	crée ou recrée la cible d'un port
/read	ouverture en mode lecture seule
/write	open for write only operation
/no-wait	pas d'attente pour les données
/skip	saute une partie des données
/allow	définit les attributs des fichiers

/custom	permet des raffinements spéciaux

Fermeture d'un port

L'accès à un port se termine quand la fonction **close** est invoquée.

Toutes les données en mémoire qui n'ont pas été sauvées seront écrites dans le fichier cible.

L'exemple ci-dessous ferme le port *fp* utilisé précedemment.

```
close fp
```

Si vous tentez de fermer un port qui n'est pas ouvert, une erreur se produira.

Un port qui a été fermé peut être réouvert, avec la fonction **open** :

```
open fp
```

Lecture du contenu d'un port

La fonction **copy** (utilisée pour les séries) va servir à lire les données d'un port, une fois celui-ci ouvert :

```
print copy fp
I wanted the gold, and I sought it,I scrabbled and mucked like
a slave....
```

Cette fonction attend normalement les données du port.

Si vous ne voulez pas attendre la fin des données, ouvrez le port avec le raffinement **/no-wait**.

Et pour lire une partie seulement des données, utilisez **copy/part** :

```
print copy/part fp 35
I wanted the gold, and I sought it,
```

Notez que le deuxième argument de **copy** peut être soit une longueur, soit une position au sein de la série, le contenu du port.

Pour lire juste une partie des données du port, vous pouvez utiliser les fonctions **find** et **copy**.

```
a: find fp "famine"
print copy/part a find a newline
famine or scurvy -- I fought it;
```

Les fonctions ordinales comme **first**, ou de position comme **next**, peuvent aussi être utilisées sur le

port :

```
print first fp
I
print first next next fp
w
```

La fonction **copy** retournera **none** lorsque toutes les données du port auront été lues.

En mode **/no-wait**, la fonction **copy** renverra une *chaîne vide* si aucune donnée n'est disponible avec le port.

```
tp: open/direct/binary/no-wait tcp://system:8000
content: make binary! 1000
while [wait tp  data: copy tp] [append content data]
close tp
```

Ecriture dans un port

La fonction **insert** sert pour l'écriture de données dans un port.

```
insert fp "I was a fool to seek it."
```

Si le port est en mode "*buffered*", la modification sera répercutée seulement lorsque le port sera fermé ou mis à jour (avec la fonction **update**). Si le port est ouvert avec **/direct**, alors tout changement est immédiatement pris en compte.

Tous les raffinements de la fonction **insert** peuvent être utilisés sur le port.

Par exemple, l'écriture de 20 caractères "espace" dans le port (avec **/dup**) :

```
insert/dup fp " " 20
```

Vous pouvez aussi utiliser sur le port les fonctions habituelles de modifications des séries, comme **remove**, **clear**, **change**, **append**, **replace**, etc.

Pour effacer un seul caractère, ou bien plusieurs :

```
remove fp
```

```
remove/part fp 20
```

Et pour effacer tous les caractères restants, écrivez :

```
clear fp
```

Mise à jour d'un port

La fonction **update** force un port à rafraîchir son état selon le fonctionnement du périphérique externe (ici fichier).

Par exemple, lors de l'écriture d'un fichier présent en cache, la fonction **update** permet de forcer le vidage du buffer.

En lecture, la fonction **update** peut être utilisée pour être sûr que toutes les données en attente ont été mises en buffer.

```
update fp
```

Attente sur un port

La fonction **wait** est essentielle aux programmes gérant de manière asynchrone des échanges de données.

Avec **wait**, vous pouvez attendre des données d'un ou plusieurs ports, ou bien qu'un time-out se produise.

La fonction peut accepter en argument un port unique :

```
wait port
```

mais un **bloc de plusieurs ports** peut aussi être fourni :

```
wait [port1 port2 port3]
```

De plus, une valeur de time-out peut également être indiquée, sous la forme d'un nombre de secondes ou comme une valeur de type **time !** :

wait [port1 port2 10]

wait [port1 port2 0:00:05]

Le premier exemple indique un time-out de 10 secondes. Le second exemple sera en time-out dans 5 secondes.

La fonction **wait** renvoie le premier port qui est prêt ou **none** si un time-out s'est produit.

```
ready: wait [port1 port2 10]
if ready [data: copy ready]
```

L'exemple précédent lira des données du premier port prêt, si un time-out ne se produit pas avant.

Pour obtenir un bloc comprenant tous les ports prêts, utilisez le raffinement **/all** :

```
ready: wait/all [port1 port2 10]
if ready [
    foreach port ready [
    append data copy port
    ]
]
```

Cet exemple ajoutera les données de tous les ports disponibles à une seule série, appelée ici *data*.

Vous pouvez aussi utiliser la fonction **dispatch** pour évaluer un bloc, ou une fonction basée sur les résultats de **wait** vis-à-vis de plusieurs ports.

```
dispatch [
    port1 [print "port1 awake"]
    port2 [print "port2 awake"]
    10 [print "time-out!"]
]
```

Usage de /no-wait et /direct

Pour utilisez **wait** avec la plupart des ports, vous aurez besoin de spécifier les raffinements **/no-wait** et **/direct** avec **open**. Ceci permet d'indiquer que les fonctions habituelles pour l'accès aux données ne doivent pas être bloquantes et que les données ne sont pas mises en cache (*buffered mode*).

```
port1: open/no-wait/direct tcp://system:8000
```

Autres modes pour un port

Mode ligne

La fonction **open** permet un accès en mode ligne.

Dans ce mode, la fonction **first** retournera une ligne de texte, plutôt qu'un caractère. (NdT : par exemple, avec un fichier texte séquentiel).

L'exemple ci-dessous lit un fichier une ligne à la fois :

```
fp: open/lines %file.txt
print first fp
I wanted the gold, and I got it --
print third fp
Yet somehow life's not what I thought it,
```

Le raffinement **/lines** est aussi utile pour les protocoles Internet qui sont orientés "lignes".

```
tp: open/lines tcp://server:8000
```

```
print first tp
```

Ecriture et lecture seules

Vous pouvez utiliser le raffinement **/read** pour ouvrir un port en lecture seule :

```
fp: open/read %file.txt
```

Les modifications faites en cache ne sont pas répercutées au fichier.

Pour ouvrir un port en écriture seulement, utilisez le raffinement **/write** :

```
fp: open/write %file.txt
```

Les ports de type fichiers ouverts avec le raffinement **/write** ne liront pas les données courantes à l'ouverture du port.

La fermeture, ou la mise à jour d'un port en écriture seule force les données existantes dans le fichier à être remplacées (écrasées).

```
insert fp "This is the law of the Yukon..."
close fp
print read %file.txt
This is the law of the Yukon...
```

Accès en mode direct

Le raffinement **/direct** ouvre un port sans mise en buffer (mémoire) des données.

C'est assez pratique pour accèder à des fichiers par morceaux, notamment lorsque le fichier est trop grand pour être stocké en mémoire.

```
fp: open/direct %file.txt
```

La lecture des données avec **copy** entraîne la modification de la position de la tête de la série, pour le port :

```
print copy/part fp 40
I wanted the gold, and I sought it,^/ I
print copy/part fp 40
scrabbled and mucked like a slave.^/Was i
```

En mode direct, le port sera toujours sur la position de tête :

```
print head? fp
true
```

La fonction **copy** renverra **none** lorsque la fin des données du port est atteinte.

Voici un exemple qui utilise des ports en mode direct pour copier un fichier quelle que soit sa taille.

```
from-port: open/direct %a-file.jpg
to-port: open/direct %a-file.jpg
while [data: copy/part from-port 100000 ][
    append to-port data
]
close from-port
close to-port
```

Sauter des données

Il existe deux façons de sauter les données existantes dans le port.

Premièrement, vous pouvez ouvrir le port avec le raffinement **/skip**. La fonction **open** passera automatiquement dans le port au point spécifié.

Par exemple :

```
fp: open/direct/skip %file.big 1000000
```

```
fp: open/skip http://www.example.com/bigfile.dat 100000
```

Deuxièmement, vous pouvez utiliser la fonction **skip** sur le port. Pour les fichiers qui sont ouverts avec les raffinements **/direct** et **/binary**, l'opération de saut est identique à l'opération seek (recherche/déplacement dans un fichier)

Les données ne sont pas lues dans le cache. Ce n'est pas possible dans le mode **/string** car les sauts de lignes interférent avec la valeur de saut.

```
fp: open/direct/binary %file.dat
fp: skip fp 100000
```

Permissions d'accès sur les fichiers

Les fichiers créés par REBOL prennent des permissions d'accès par défaut. Sur les systèmes d'exploitation Windows et Macintosh, les fichiers sont créés avec des privilèges permettant leur contrôle total. Sur les systèmes Unix, les fichiers sont créés avec des permissions selon l'*umask* courant.

Lorsqu'on utilise **open** ou **write** pour accèder à un fichier, le raffinement **/allow** peut être utilisé pour définir les permissions d'accès.

Le raffinement **/allow** prend un bloc en argument. Ce bloc peut être constitué de n'importe lequel ou de tous les mots : **read**, **write**, **execute**.

Restrictions liées au système d'exploitation

Le raffinement **/allow** permettra de définir des permissions d'accès uniquement sur les systèmes

d'exploitation qui le permettent. Si le système ne supporte pas l'attribution de certaines permissions, elles seront ignorées. Par exemple, les fichiers sur les systèmes Unix peuvent être défini comme exécutables (execute), mais les systèmes Windows et Macintosh ne supportent pas cette option.

Pour un système Unix, les possibilités de permissions de fichiers sont restreintes à celles de l'utilisateur.

Suivant le système, l'usage d'**/allow** conduit à effacer les permissions d'accès pour les utilisateurs.

Pour mettre un fichier en lecture seule, utilisez **open/allow**, ou **write/allow** avec un bloc read :

```
write/allow %file.txt [read]
```

Pour donner les droits en lecture et en exécution à un fichier :

```
open/allow %file.txt [read execute]
```

Vous pouvez définir des droits similaires, en écriture :

```
write/allow %file.txt [read write]
```

Pour supprimer tous les accès à un fichier (pour les systèmes d'exploitation qui font la différence), mettez un bloc vide pour les permissions :

```
write/allow %file.txt []
```

Pour un accès complet :

```
write/allow %file [read write execute]
```

Ports relatifs aux répertoires

Les ports relatifs aux répertoires vous permettent d'accèder directement à ceux-ci.

Quand vous ouvrez un répertoire, vous obtenez l'accés au répertoire présenté comme un bloc de noms de fichiers.

```
mydir: open %intro/
forall mydir [print first mydir]
CVS/
history.t
intro.t
overview.t
quick.t
close mydir
```

Vous pouvez avancer à une position spécifique à l'intérieur de la série du répertoire et effacer un fichier, avec par exemple le code suivant :

```
dir: open %.
remove next dir
close dir
```

Ce code efface le second fichier dans le répertoire courant. De la même manière,

```
remove at dir 5
```

devrait effacer le cinquiéme fichier dans le répertoire, et :

```
clear dir
```

devrait effacer tous les fichiers du répertoire.

Pour effacer tous les fichiers dont le nom contient "junk", vous pouvez écrire :

```
dir: open %intro/
while [not tail? dir] [
    either find first dir "junk" [remove dir][
        dir: next dir
    ]
]
close dir
```

Les modifications apportées au répertoire sont répercutées quand le port relatif au répertoire est fermé ou quand il est mis à jour.

Pour forcer l'application d'un changement, utilisez le code suivant :

```
update dir
```

NdT : il semblerait que **update** ne fonctionne pas, avec le scheme 'directory.

La méthode d'accès à un répertoire peut aussi être utilisée pour changer les noms des fichiers. Après l'ouverture du port, la ligne :

```
change at dir 3 %newname.txt
```

permet de renommer le troisième fichier dans le répertoire. Pareillement, le nom de n'importe lequel des fichiers dans le répertoire peut être modifié.

Voici un exemple pour renommer tous les fichiers dans un répertoire en ajoutant le mot REBOL au nom initial :

```
dir: open %intro/
forall dir [insert first dir "REBOL"]
close dir
```

NdT : l'exemple ne fonctionne pas. Bug signalé à RT.

Updated 6-Apr-2005 - Copyright REBOL Technologies - Formatted with MakeDoc2 Translation by Philippe Le Goff