The REBOL Documentation Project

-- FR - Documentation REBOL - Articles Techniques --

Articles Techniques

Guide du développeur REBOL/Services

Philippe Le Goff

Première publication : 27 mai 2006, et mis en ligne le samedi 27 mai 2006

Résumé :

Ce document est la traduction française du Rebol/Services Start, le guide du développeur.

Traducteur : Philippe Le Goff

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
jeudi 6 avril 2006	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Introduction

Ce document a été écrit pour ceux d'entre vous qui veulent comprendre les REBOL/Services et comment les utiliser, mais n'ont pas le temps de lire des dizaines de pages de documentation technique. Les informations présentées ici devraient être suffisantes pour vous permettre de démarrer avec vos propres scripts et applications, mais les utilisateurs expérimentés auront intérêt à lire

certains autres documents techniques concernant les REBOL/Services.

2.1 Que sont les REBOL/Services ?

Il serait facile juste de se lancer et de démarrer dés à présent avec les REBOL/Services, mais avant de faire cela, vous devriez acquérir un minimum de compréhension de ce qu'ils sont et de comment ils fonctionnent.

Les REBOL/Services fournissent un moyen simple, élégant de partager de l'information entre les programmes informatiques. Ils peuvent être utilisés pour échanger de l'information entre des clients et des serveurs qui peuvent être éloignés de milliers de kilomètres, ou simplement entre applications fonctionnant localement sur votre ordinateur.

Les REBOL/Services implémentent un concept appelé "Service Orienté Architecture" ou SOA pour faire court. L'idée de base d'un SOA est que vous envoyez un message, une requête à un autre programme ("un service") qui va tenter de répondre à la requête et de renvoyer le résultat.

2.2 Avantages des Services

Les premiers avantages des REBOL/Services sont les suivants :

Facilité d'emploi : *Vous pouvez accéder au service, ou implémenter un service personnalisé avec une seule ligne de code*

Sécurité : *L'authentification forte et le cryptage des données sont pré-inclus, les commandes sont émises en utilisant l'approche par dialecte spécifique à REBOL (aucune exécution directe de fonction ne se produit). Pour des systèmes extrêmement sécurisés, des clés privées peuvent être utilisées, ainsi le fait de connaître le nom et le mot de passe d'un utilisateur n'est pas suffisant.*

Flexibilité : *Les REBOL/Services sont basés sur le concept propre à REBOL, de dialecte (dialecting) qui permet un grand degré de liberté et de contrôle avec un minimum de code. De plus, les services peuvent être utilisés au-dessus de divers protocoles de transports (TCP et HTTP sont fournis en standard).*

Fiabilité : *Plus votre code est court, plus votre application est fiable. Lorsque votre code fait seulement quelques Kos, comparé à quelques Mos, il y a de bonnes probabilités d'éliminer tous les bogues.*

Commodité : *Chaque version de REBOL aura cette technologie incluse en standard, de sorte que vos scripts et vos applications peuvent tirer parti d'elle sans ajouter de librairies dédiées.*

Un standard ouvert : *Pour permettre aux développeurs d'apporter des améliorations et des suggestions, tout comme de fournir des jeux de services particuliers.*

Une liste plus détaillée des avantages qu'apportent les REBOL/Services peut être trouvée par ailleurs.

2.3 Types de Services

En tant que SOA, les REBOL/Services vous offrent un moyen facile de créer une grande diversité d'applications telles que :

- Partage sécurisé de fichiers
- Administration à distance de serveurs
- Gestion de code source
- Collaboration sur des documents
- Mise à jour de site Web
- Echange de messages (incluant l'IM, l'Instant messaging)
- Gestion de tâches
- Traitement de cartes de crédits
- Services d'horodatage, de synchronisation.
- Vote et participation à un scrutin
- Allocation de ressources systèmes
- Application de comptabilité
- Constitution de tableaux de bords

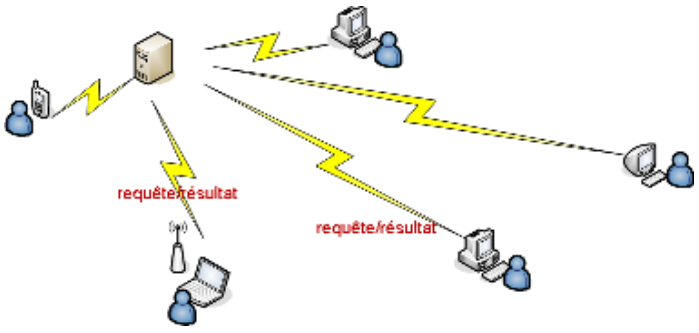
Il n'y a pas réellement de fin à cette liste, car cela prend seulement une page ou deux pour implémenter la plupart des services. Je suis certain que vous saurez imaginer encore d'autres usages.

2.4 Pour plus d'information

Beaucoup d'autres informations à propos des REBOL/Services seront documentées dans les mois à venir. Nous fournirons un lien vers ces informations sur notre site Web.

3 Comment fonctionnent les REBOL/Services ?

Pour utiliser un service, vous envoyez simplement une requête. Le service traite la requête et retourne un résultat. Le service peut être n'importe où. Il peut se trouver sur un serveur distant, ou sur un autre client dans votre réseau locale, ou être un autre processus sur la même machine. Il n'y a pas de différences.



3.1 La séquence entre la requête et le résultat

Regardons plus attentivement ce qu'il se passe :

Requête : *La requête est émise vers le service. Le format de la requête est très simple : c'est un bloc REBOL. Le bloc commence avec un mot (indiquant la commande) et contient d'autres mots et des valeurs qui sont structurés dans l'ordre où le service les attend. Il s'agit d'un dialecte en REBOL, qui offre tous les avantages de ce concept.*

Traitement : *La requête est traitée par le service. Il doit analyser la requête (la commande), agir en conséquence, et mettre en forme le résultat à renvoyer. Tout cela est habituellement déterminé par la fonction PARSE, mais d'autres méthodes sont également autorisées.*

Résultat : *Lorsque le traitement est complet, le résultat est retourné vers le client. Le format du résultat est quelque chose d'assez similaire à celui de la requête, mais pas identique.Le format permet au client de connaître rapidement si la requête a été un succès et d'obtenir les valeurs résultant du traitement.*

Tous les transferts depuis et vers le service sont cryptés. Divers niveaux de cryptage sont possibles. Ils sont décrits dans un autre document.

3.2 Exemples de requêtes

A quoi ressemble le bloc d'une requête ? Cela dépend du service. Par exemple, la requête peut être aussi simple que ce bloc :

```
[date]
```

La requête demande la date courante au service. Cette commande fait partie des commandes fournies en standard pour tous les services. Le bloc d'une requête plus complexe pourrait être comme ceci :

```
[file/get %maui.jpg 12-Mar-2005/10:00 %hawaii.jpg 14-Mar-2005/2:20]
```

Ce bloc demande le transfert de deux fichiers vers le client, si les fichiers sont plus récents que les dates indiquées. Si les fichiers sont plus anciens, ils ne sont pas récupérés.

Une requête peut aussi inclure plusieurs blocs de commandes. Voici trois commandes émises dans la même requête :

```
[date]
[info title]
[file/put %photo.jpg (read/binary %photo.jpg)]
```

Le résultat qui sera retourné au client contiendra trois blocs de résultats.

3.3 Exemples de résultats

Les résultats renvoyés par un service sont constitués de deux parties :

un indicateur de succès,
et un bloc qui comprend les valeurs résultant du traitement.

Par exemple, la commande **date** ci-dessus va retourner :

```
ok [date 25-Mar-2005/12:08:12-8:00]
```

L'indicateur **ok** signifie que le traitement de la commande s'est correctement déroulé. S'il échoue, pour une raison quelconque, le mot **fail** est renvoyé et si une erreur se produit, le mot **error** est retourné.

Le bloc qui suit l'indicateur de succès rappelle la commande et vous donne son résultat. Ce bloc est aussi exprimé dans un dialecte, car le format du bloc dépend du service. Cependant, le premier item dans le bloc est toujours la commande. (Ceci rend aisé pour des clients complexes la possibilité de générer des événements en callback, au sein d'une application.)

Pour des commandes multiples dans vos requêtes, vous obtiendrez de multiples commandes dans vos résultats. L'exemple du paragraphe précédent conduira à ces résultats :

```
ok [date 25-Mar-2005/12:08:12-8:00]
ok [info "Default REBOL Service"]
ok [file/put %photo.jpg 20456]
```

Il y a une chose supplémentaire à savoir à propos du bloc de résultat et que nous n'avons pas montré ci-dessus (pour simplifier l'exemple).

Il contient un indicateur et un bloc récapitulatifs de la requête. Un exemple de récapitulatif ressemble à ceci :

```
done [reply seq 1 service "Generic Service" commands 1 time 0:00:01]
```

Le mot **done** indique qu'aucune erreur ne s'est produite et que la requête a été complètement traitée. Le suivi du succès de la requête en est facilité, même si elle contient de multiples commandes. Le bloc contient une information qui récapitule l'état du traitement de la requête. La plupart du temps, vous pourrez ignorez sans souci cette information.

4 Accéder à un service

Bon, à présent, voyons comment accéder à un REBOL/Service. Les services sont implémentés en REBOL au travers d'une interface basée sur des fonctions (function-based interface). Cette approche permet de conserver une interface très simple, en minimisant ce que l'utilisateur à besoin de savoir pour faire son travail. D'autre part, sous la surface visible, un mécanisme de port asynchrone en REBOL est utilisé, permettant aux développeurs expérimentés un très grand niveau de contrôle.

4.1 Le plus simple exemple

Pour débuter, le mieux est de regarder une ligne de code qui accède à un service :

```
result: do-service tcp://server:8000 [date]
```

Cette ligne envoie une requête **date**, attend la réponse, et renvoie le résultat. On appelle ceci une requête synchrone. La fonction **do-service** va attendre, jusqu'au moment où le serveur a fini son traitement, avant de retourner le résultat.

4.2 Accès asynchrone (No-wait)

Pour certains types de programmes, vous voudrez peut-être utiliser une requête asynchrone. Dans ce cas, la fonction n'attend pas le résultat . Vous pouvez soit fournir une fonction qui sera appelée lorsque le résultat sera reçu (fonction en callback), soit attendre la réponse.

Voici un exemple de requête asynchrone :

```
send-service/action tcp://server:8000 [date] [print mold result]
```

Ici, une requête **date** est émise vers le service, mais sans attendre le résultat. Lorsque le résultat est reçu, le second bloc est évalué, pour afficher le résultat. Cette approche est très similaire avec le principe des interfaces graphiques, qui sont par nature asynchrones (plusieurs choses réalisées en même temps). S'il est nécessaire que vous attendiez le résultat d'une requête faite précédemment, vous pouvez utiliser la fonction **wait-service**.

Voici un exemple :

```
req: send-service tcp://server:8000 [date]
do-some-other-stuff
result: wait-service req
```

La requête est envoyée, puis tandis qu'elle est traitée, vous pouvez faire d'autres choses. Lorsque vous avez besoin du résultat, vous pouvez l'attendre et le placer dans la variable result.

4.3 Emissions de requêtes multiples

L'exemple ci-dessus illustre comment utiliser une URL pour appeler un service. Cependant, la plupart du temps, dans les applications réelles, vous n'aurez pas à fournir une URL à chaque fois. Au lieu de cela, vous établirez une connexion qui restera active pour de multiples requêtes. Ceci peut être réalisé avec la fonction **open-service** :

```
port: open-service tcp://server:8000
```

Elle renvoie un port qui peut être utilisé comme argument pour d'autres fonctions :

```
result: do-service port [date]
```

```
send-service/action port [date] [print mold result]
```

Lorsque vous aurez fini, vous enverrez un ordre pour fermer la connexion :

```
close-service port
```

Il s'avère également que vous pouvez employer les fonctions **send-service** et **login-service** pour ouvrir le port et pour le maintenir ouvert. Il n'est pas nécessaire d'utiliser uniquement **open-service** ; cependant, **open-service** permet de spécificier des options particulières, susceptibles d'être employées pour la connexion.

4.4 Les fonctions utiles pour les requêtes à un service

Voici un résumé des fonctions qui sont utilisées pour manipuler les requêtes émises vers un service. Elles sont implémentées dans l'API client (Application Programming Interface).

Function	Description
do-service	Émet une requête vers un service et attend le résultat. Renvoie le résultat. Le résultat sera simplifié si c'est possible (Les champs de la réponse pourront ne pas être tous renvoyés. Voir plus loin.)
send-service	Émet une requête vers un service mais n'attend pas le résultat. Renvoie un objet req-message qui peut être utilisé par d'autres fonctions listées ci-dessus. Un bloc (ou une fonction) peut être aussi spécifié en option pour le callback, en utilisant le raffinement /action.
wait-service	Attend le résultat d'une requête émise précédemment avec send-request et renvoie le résultat. Le résultat n'est pas simplifié (il contient tous les champs de la réponse faite par le service.)
query-service	Retourne le résultat d'une requête émise précédemment avec send-request, si elle a été reçue. Sinon, retourne NONE. Cette fonction vous permet de tester la disponibilité d'un résultat sans l'attendre. (NdT : une sorte de "polling" sur la réponse, en quelque sorte)
open-service	Ouvre une connexion à un service et renvoie un port REBOL, qui peut être utilisé avec toutes les autres fonctions de l'API. Permet de spécifier des options supplémentaires relatives au service, comme des délais pour des time-outs, ou des clés, des méthodes de cryptage.
close-service	Ferme le port lié au service ouvert précédemment. Aucune erreur ne se produit si le service a déjà été fermé.
abort-service	Interrompt une requête send-service précédente, si c'est possible. Renvoie TRUE si la fonction s'est exécutée correctement. Renvoie FALSE si la requête a déjà été envoyée au service et ne peut être interrompue.
login-service	Il s'agit d'une fonction qui facilite l'authentification lors de l'accès à un service et initie une session complètement cryptée. Cette fonction ne rend la main que lorsque l'étape de login est complète (l'implémentation est faite en tant que fonction synchrone). Le login asynchrone est implémenté par l'envoi d'une requête login au service avec la fonction send-service.
logout-service	Termine une session authentifiée et cryptée.

5 Requêtes à un service

Une requête à un service est un bloc de commande(s) qui est émis vers le service par les fonctions **do-service** et **send-service**.

5.1 Requêtes uniques

Pour conserver une certaine simplicité aux exemples ci-dessous, seule la requête **date** est présentée :

```
result: do-service tcp://server:8000 [date]
```

Les requêtes uniques peuvent être émises ainsi dans un bloc. Ici, le mot **date** est la commande demandée, et elle ne nécessite pas d'arguments. D'autres commandes peuvent nécessiter des arguments additionnels. Ceux-ci peuvent également être fournis au sein du bloc :

```
result: do-service tcp://server:8000 [info title]
```

```
result: do-service tcp://server:8000 [login "carl"]
```

Les fonctions de requête à un service effectuent un COMPOSE/deep sur le bloc. Ceci vous permet d'insérer des termes à évaluer à l'intérieur du bloc. Par exemple :

```
result: do-service tcp://server:8000 [
    file/put %photo.jpg (read/binary %photo.jpg)
]
```

Dans cet exemple, le fichier photo.jpg est lu comme un fichier binaire, puis est inséré dans la requête avant son envoi.

5.2 Requêtes multiples

Le format des requêtes à commande unique permet de conserver un code simple. La forme la plus générale permet cependant l'envoi de commandes multiples à un service dans une seule requête. Pour cela, chaque commande doit être encapsulée dans un sous-bloc :

```
result: do-service tcp://server:8000 [
    [date]
    [info title]
    [file/put %photo.jpg (read/binary %photo.jpg)]
    [file/get %manual.txt]
]
```

Cet exemple envoie les quatre commandes via un service avec une seule requête. Les résultats retournés seront dans un format similaire, voir la section "Résultats de services" plus loin.

L'écriture utilisée (paths) pour les commandes file/put et file/get indique qu'elles sont trouvées dans le contexte du REBOL/Service **file** (le service de fichier), pas dans le contexte global par défaut. Voir la section "Contexte d'un service" ci-dessous.

5.3 Les requêtes utilisent un dialecte

Il est important de comprendre que les requêtes faites aux REBOL/Services sont des dialectes REBOL. Ceci étant, elles n'appellent pas les fonctions REBOL directement au sein du service, mais sont interprétées comme un sous-langage spécifique à un domaine.

Cette approche :

- Permet une meilleure sécurité car l'API du langage REBOL n'est jamais directement utilisée.
- Permet d'avoir une plus grande diversité d'expression qu'en RPC (remote procedure call) ou RMI (remote method invocation). Les jeux de commandes peuvent être des sous-langages complets.
- Nécessite très peu de code pour implémenter des services en comparaison d'autres approches.
- Les rend plus facile à tester. Les dialectes de commandes sont manipulés par la fonction PARSE qui être facilement appelée et déboguée de manière autonome durant leur développement. Le service n'a pas besoin d'être mis en ligne pour être massivement testé.

6 Résultats des Services

Le résultat d'un service est une ou plusieurs valeurs retournées par le traitement de la requête.

6.1 Résultats simples et uniques

La fonction **do-service** fournit une simplification des résultats pour des requêtes particulières. Ceci se voit dans les exemples suivants :

```
print mold do-service tcp://server:8000 [date]
25-Mar-2005/12:08:12-8:00
print mold do-service tcp://server:8000 [info title]
"Default REBOL Service"
```

Dans la plupart des cas, le résultat est retourné sous la forme d'une valeur unique. Ceci est uniquement vrai pour le cas où **do-service** est utilisé avec une seule commande. Les requêtes à plusieurs commandes renvoient des blocs à plusieurs résultats, comme décrit ci-dessous.

Note de conception : Nous pouvons être amenés à réévaluer cette façon d'opérer. Elle est similaire dans le comportement à la fonction LOAD, laquelle si elle n'est pas bien connue par l'utilisateur, peut conduire à des erreurs. Me contacter pour donner votre avis sur ce point.

6.2 Résultats multiples

Lorsque des commandes multiples sont envoyées à un service dans une seule requête, leurs résultats sont retournés sous forme de plusieurs blocs. Voici un exemple de résultat issu d'une requête à plusieurs commandes.

```
probe do-service tcp://server:8000 [
    [date]
    [info title]
    [file/put %photo.jpg (read/binary %photo.jpg)]
    [file/get %manual.txt]
]
[
    ok [date 25-Mar-2005/12:08:12-8:00]
    ok [info "Default REBOL Service"]
    ok [file/put %photo.jpg 20456]
    ok [file/get %manual.txt 25-Mar-2005 #{...}]
```

```
]
```

Chaque résultat commence avec un indicateur de succès (**ok**), suivi d'un bloc, le contenu du résultat. Pour rendre plus facile l'identification des résultats, ceux-ci sont toujours retournés **dans le même ordre** que les commandes, et chaque résultat inclut la commande requise : c'est le premier item du bloc. Il vous est possible, selon les options particulières choisies, d'avoir des résultats mixtes comprenant des requêtes réussies et d'autres qui ont échouées :

```
[
    ok [date 25-Mar-2005/12:08:12-8:00]
    ok [info "Default REBOL Service"]
    fail [file/put not-allowed]
    fail [file/get not-exists]
]
```

De plus, le bloc de résultat peut inclure un en-tête optionnel qui récapitule les résultats de la requête, en comprenant d'autres détails qui sortent du cadre de ce document.

6.3 Contextes propres aux commandes des REBOL/Services

Un serveur peut incorporer de multiples contextes pour les commandes des services. Chaque contexte fournit un lot de commandes pour un service spécifique. Par exemple, le serveur par défaut (pré inclus) fournit les contextes suivants :

Home - implémente des commandes qui font partie de tous les services. C'est le contexte par défaut et c'est un service requis. Son principal objet est de fournir des informations sur un serveur.

Admin - fournit les commandes pour contrôler votre service, sa mise à jour, redémarrage, gestion des utilisateurs, récupération des logs et d'autres encore.

File - Un ensemble de commande pour accéder ou transférer de petits fichiers (moins de quelques Mos). Ce service peut être utilisé pour uploader ou télécharger des pages web, des graphiques, du code et d'autres fichiers.

Les commandes au sein d'une requête peuvent explicitement spécifier un contexte en utilisant la notation avec les paths :

```
[admin/reset]
[file/get %photo.jpg]
```

(NdT : on a une notation du type /)

Ou, vous pouvez sélectionner un contexte différent avec la commande SERVICE :

```
[service admin]
[add-user "Bob" "Robert Smith" ...]
[change-user 47 user "Jenny"]
```

Ici les commandes add-user et change-user font partie du contexte admin. Vous pouvez changer le contexte de la commande autant de fois que vous le voulez au sein de la même requête. De plus, les commandes du contexte home sont toujours utilisables au sein des autres contextes, tant qu'elles n'ont pas été écrasées par des commandes dans le contexte courant. par exemple, ceci est valide :

```
[service admin]
[add-user "Bob" "Robert Smith" ...]
[change-user 47 user "Jenny"]
[date]
```

Ici la commande date fait partie du service (contexte) home, pas du service (contexte) admin.

7 Créer un service

A présent que vous savez comment accéder aux REBOL/Services, voici quelques bases pour créer un service.

7.1 Le plus simple serveur

Cet exemple démarre un serveur, mais n'appelle aucun service spécifique. Il n'utilisera que les services standards (home, admin, et file, comme indiqué précédemment.)

```
service: start-service tcp://:8000
```

Le serveur utilise une connexion directe en TCP sur le port 8000. Immédiatement, il commencera à traiter des requêtes.

Voici un exemple de ce que vous pouvez saisir dans un script et tester :

```
REBOL [Title: "Example Server"]
service: start-service tcp://:8000
ask "PRESS ENTER TO QUIT"
wait []         stop-service service
```

Le serveur continuera à fonctionner jusqu'à ce que vous pressiez la touche "Enter". Lorsque celle-ci est activée, le service s'arrête.

7.2 Créer votre propre service

Pour créer votre propre service, tout ce que vous aurez à faire est de créer un contexte de service avec un dialecte pour vos commandes.

Un contexte de service est un objet REBOL qui est mis dans un fichier, ce fichier étant chargé lorsque votre serveur démarre. Voici un exemple de service qui implémente un petit système de bulletins

électroniques (Bulletin board).

```
name: 'bbs-example
title: "Micro-BBS Service"
description: " un BBS très simplifié."

; Message format: [date author message]
messages: any [attempt [load %messages.r] copy []]

commands: [
    'put "Store a new message"
        arg: string! "Author" string! "Message" (
            write/append messages mold reduce [now arg/1 arg/2]
            result: true
        )
    | 'get "Get a messages by number"
        (result: copy [])
        some [
            arg: integer!
            (repend result [arg/1 pick messages arg/1])
        ]
    | 'list "List new message numbers since a given date or number"
        (result: copy []) [
            arg: date! (
                num: 1
                foreach msg messages [
                    if msg/1 >= arg/1 [append result num]
                    num: num + 1
                ]
            )
            | arg: integer! [
                repeat n length? skip messages arg/1 [
                    append result n
                ]
            ]
        ]
    ]
]
```

Notez que les chaînes de caractères à l'intérieur du dialecte pour les commandes sont utilisées pour documenter le code, et elles sont extraites lorsque le service est initialisé. Le bloc de règles passé à la fonction PARSE ne contient pas de chaînes littérales. Il existe aussi une notation spéciale à utiliser pour inclure des chaînes spécifiques à une langue. (Plus d'information plus tard sur ce point). Les variables result et arg sont définies localement dans la fonction là où le bloc de la commande est évalué. La variable result contient la valeur ou le bloc qui sont retournés depuis la commande et renvoyés au client.

Pour mettre le service en ligne, vous devez l'indiquer dans le bloc des services qui est fourni en tant qu'option à la fonction start-service.

```
service: start-service/options tcp://:8000 [
    services: [%bbs-example.r]
]
```

Il est également possible d'ajouter le service dans la configuration du serveur en utilisant la fonction handle-service, mais cette caractéristique sera décrite dans un document à part.