The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 13 - Protocoles Réseau

Philippe Le Goff

Première publication : 19 octobre 2005, et
mis en ligne le mercredi 19 octobre 2005

Résumé :

Ce document est la traduction française du Chapitre 13 du User Guide de REBOL/Core, qui concerne les protocoles Réseau.

Ce document est la traduction française du Chapitre 13 du User Guide de REBOL/Core, qui concerne les protocoles Réseau.

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
5 juin 2005 21:02	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Présentation

REBOL inclut en standard plusieurs des plus importants protocoles Internet.

Ces protocoles sont faciles à utiliser au sein de vos scripts ; ils ne requièrent aucune librairie ou fichier à inclure, et la plupart des opérations peuvent être réalisées en une seule ligne de code.

La liste ci-dessous indique les protocoles réseau supportés :

DNS : *Domain Name Service : traduit les noms d'ordinateur en adresses IP et vice-versa.*

Finger : *Permet d'obtenir des informations sur des utilisateurs, par leurs profils.*

Whois : *Récupére des informations sur l'enregistrement d'un domaine (domain registration).*

Daytime : *Network Time Protocol : permet d'obtenir l'heure depuis un serveur.*

HTTP : *Hypertext Transfer Protocol. Utilisé pour le Web.*

SMTP : *Simple Mail Transfer Protocol. Utilisé pour l'envoi d'e-mails.*

POP : *Post Office Protocol. Utilisé pour récupérer des mails.*

FTP : *File Transfer Protocol. Transfert de fichiers avec un serveur.*

NNTP : *Network News Transfer Protocol. Pour émettre ou lire des news Usenet.*

TCP : *Transmission Control Protocol. Protocole de base d'Internet.*

UDP : *User Datagram Protocol. Protocole non orienté connexion basé sur l'envoi de datagrammes.*

De plus, vous pouvez créer des agents pour d'autres protocoles Internet ou créer votre propre protocole.

3 Bases Réseau pour REBOL

3.1 Modes de fonctionnement

Il existe deux modes de base pour les opérations réseau : atomique ou basé sur un port.

Les opérations réseau en mode "atomique" sont celles qui sont accomplies avec une unique fonction. Par exemple, vous pouvez lire une page Web entière avec un seul appel à la fonction **read**. Il n'est pas nécessaire de séparer l'ouverture de la connexion et la lecture. Tout cela est fait automatiquement avec la fonction **read**.

Par exemple, vous pouvez saisir :

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

Le serveur cible est trouvé et ouvert, a page Web est transférée, et la connexion est fermée.

Les opérations réseau basées sur les ports sont celles qui s'appuient sur une approche traditionnelle en programmation. Elles supposent l'ouverture d'un port, et effectuent diverses opérations sur le port. Par exemple, si vous voulez lire votre courrier électronique depuis un serveur POP, message par message, vous devrez utiliser cette méthode. Voici un exemple qui lit et affiche tous vos emails.

```
pop: open pop://user:pass@mail.example.com
```

```
forall pop [print first pop]
close pop
```

L'approche atomique est plus facile, mais aussi plus limitée. L'approche basée sur la gestion des ports autorise plus d'opérations, mais suppose aussi une plus grande compréhension des aspects réseau.

3.2 Spécification de ressources réseaux

REBOL fournit deux approches pour spécifier des ressources réseau : les spécifications d'URLs et de ports.

Les URLs (Uniform Resource Locators) sont utilisées pour identifier une ressource réseau, comme une page Web, un site FTP, une adresse email, un fichier, une autre ressource ou un service. Les URLs sont un type de données intrinsèque à REBOL, et elles peuvent être exprimées directement dans le langage.

La notation standard pour les URLs consiste à écrire le "scheme" (NdT : souvent le protocole), suivi de sa spécification :

```
scheme:specification
```

Le scheme est souvent le nom du protocole, tel que HTTP, FTP, SMTP, et POP ; d'autre part, ce n'est pas une nécessité. Un "scheme" peut être n'importe quel nom qui identifie la méthode utilisée pour accéder à une ressource.

Le format de la spécification relative à un "scheme" dépend de celui-ci ; cependant, la plupart des schemes partagent un format commun pour identifier les serveurs réseau, les noms d'utilisateur, les mots de passe, les numéros de ports, et les chemins vers les fichiers. Voici quelques formats couramment utilisés :

```
scheme://host

scheme://host:port

scheme://user@host

scheme://user:pass@host

scheme://user:pass@host:port

scheme://host/path

scheme://host:port/path

scheme://user@host/path

scheme://user:pass@host/path
```

```
scheme://user:pass@host:port/path
```

Voici la liste des champs utilisés dans les formats précédents (Network Resource Specification).

scheme : *Le nom utilisé pour identifier le type de ressource, souvent le même que le protocole. Par exemple, HTTP, FTP, et POP.*

host : *Le nom réseau ou l'adresse pour une machine. Par exemple, www.rebol.com, cnn.com, accounting.*

port : *Le numéro de port de la machine cible, pour le scheme en cours. Normalement, les ports sont standardisés, donc cette information n'est pas requise la plupart du temps. Exemples : 21, 23, 80, 8000.*

user : *un nom d'utilisateur pour accéder à la ressource.*

pass : *Un mot de passe pour authentifier le nom d'utilisateur.*

path : *Un chemin de fichier ou une autre méthode pour référencer la ressource. Cette valeur dépend du scheme utilisé. Certains schemes incluent des modèles et des arguments de scripts (comme avec CGI).*

Une autre manière d'identifier une ressource est une spécification de port en REBOL. En fait, lorsqu'une URL est utilisée, elle est automatiquement convertie en spécification de port. Une spécification de port peut accepter beaucoup plus d'arguments qu'une URL, mais nécessite plusieurs lignes pour les définir.

Une spécification de port est une définition d'objet sous forme de bloc, qui fournit chacun des paramètres nécessaires pour accéder à la ressource réseau. Par exemple, l'URL pour accéder à un site Web est :

```
read http://www.rebol.com/developer.html
```

mais elle peut aussi être écrite sous la forme :

```
read [
    scheme: 'HTTP
    host: "www.rebol.com"
    target: %/developer.html
]
```

L'URL pour une ressource FTP à lire peut être :

```
read ftp://bill:vbs@ftp.example.com:8000/file.txt
```

mais elle peut aussi être écrite sous la forme :

```
read [
    scheme: 'FTP
    host: "ftp.example.com"
```

```
    port-id: 8000
    target: %/file.txt
    user: "bill"
    pass: "vbs"
]
```

De plus, il y a beaucoup d'autres champs pour le port qui peuvent être mentionnés, comme la durée pour un time-out, le type d'accès, et la sécurité.

3.3 Schemes, Agents (handlers) et Protocoles

Le fonctionnement de REBOL pour le réseau exploite les schemes pour identifier les agents (handlers) qui communiquent avec les protocoles.

En REBOL, le scheme est utilisé pour identifier la méthode d'accès à une ressource. Cette méthode utilise un objet codé, qui est appelé un agent. Chacun des schemes, supportés par REBOL, pour une URL (comme HTTP, FTP) a un agent. La liste des schemes peut être obtenue avec :

```
probe next first system/schemes
[default Finger Whois Daytime SMTP POP HTTP FTP NNTP]
```

De surcroît, il existe de schemes de bas niveau qui ne sont pas indiqués ici. Par exemple, les schemes TCP et UDP sont utilisés pour les communications directes et de bas niveau.

De nouveaux schemes peuvent être ajoutés à cette liste. Par exemple, vous pouvez définir votre propre scheme, appelé FTP2, qui utilisera des caractéristiques spéciales pour l'accès FTP, comme fournir automatiquement votre nom d'utilisateur et votre mot de passe, de sorte que vous n'ayez pas à les inclure dans les URLs FTP.

La plupart des agents (handlers) sont utilisés pour fournir une interface à un protocole réseau. Un protocole est utilisé pour communiquer entre des périphériques divers, comme des clients et des serveurs.

Bien que chaque protocole soit légèrement différent dans sa façon de communiquer, il peut y avoir des choses communes avec d'autres protocoles.

Par exemple, la plupart des protocoles requièrent une connexion réseau à ouvrir, à lire, à écrire et à fermer. Ces opérations communes sont accomplies en REBOL par un agent par défaut.

Cet agent rend des protocoles comme finger, whois, et daytime presque triviaux à implémenter.

Les agents pour les schemes sont écrits sous forme d'objets. L'agent par défaut sert d'objet racine (root object) pour tous les autres agents. Quand un agent nécessite un champ particulier, comme une valeur de time-out à utiliser pour lire les données, si la valeur n'est pas définie spécifiquement dans l'agent, elle sera fournie par l'agent par défaut.

Donc, les agents surchargent l'agent par défaut, avec leurs champs et leurs valeurs. Vous pouvez aussi créer des agents qui utilisent les valeurs par défaut d'autres agents. Par exemple, vous pouvez

utiliser un agent FTP2 qui prend ses champs manquants d'abord dans l'agent FTP, puis ensuite dans l'agent par défaut.

Lorsqu'un port est utilisé pour accéder à des ressources réseau, il est lié à un agent spécifique. Ensemble, l'agent et le port forment une unité qui est utilisée pour fournir l'information sur les données, le code, et le statut permettant de traiter tous les protocoles.

Le code source pour les agents peut être obtenu à partir de l'objet **system/scheme**. Ceci peut être utile si vous voulez modifiez le comportement d'un agent ou construire le votre. Par exemple, pour visualiser le code de l'agent Whois, saisissez :

```
probe get in system/schemes 'whois
```

Notez que ce que verrez est un mélange de l'agent par défaut avec l'agent whois. Le code source actuel qui est utilisé pour créer l'agent Whois fait seulement quelques lignes :

```
make Root-Protocol [
    open-check:  [[any [port/user ""]] none]
    net-utils/net-install Whois make self [] 43
]
```

3.4 Surveillance d'agents

A des fins de déboguage, vous pouvez surveiller les actions de chaque agent. Chaque agent produit sa propre trace pour le déboguage, qui indique quelles opérations ont été réalisées. Pour mettre en route le déboguage réseau, activez-le avec la ligne suivante :

```
trace/net on
```

Pour désactiver le déboguage réseau, utilisez :

```
trace/net off
```

Voici un exemple :

```
read pop://carl:poof@zen.example.com
URL Parse: carl poof zen.example.com none none none
Net-log: ["Opening tcp for" POP]
connecting to: zen.example.com
Net-log: [none "+OK"]
Net-log: {+OK QPOP (version 2.53) at zen.example.com starting.}
Net-log: [["USER" port/user] "+OK"]
Net-log: "+OK Password required for carl."
Net-log: [["PASS" port/pass] "+OK"]
** User Error: Server error: tcp -ERR Password supplied for "carl"
is incorrect.
** Where: read pop://carl:poof@zen.example.com
```

4 Démarrage initial

Les fonctionnalités réseau REBOL sont intégrées. Pour créer des scripts qui utilisent des protocoles réseau, vous n'avez pas besoin d'inclure des fichiers spéciaux ou des librairies. Le seul pré-requis est de fournir l'information minimale nécessaire pour activer les protocoles pour atteindre les serveurs ou passer les pare-feux ou les proxys. Par exemple, pour envoyer un e-mail, le protocole SMTP nécessite un nom de serveur SMTP et une adresse e-mail de réponse.

4.1 Paramétrages de base pour le réseau

Quand vous utilisez REBOL pour la première fois, il vous sera demandé d'indiquer les paramétrages nécessaires au réseau, lesquels seront stockés dans le fichier user.r.

REBOL utilise ce fichier pour charger les paramétrages réseau nécessaires à chacun de ses démarrages. Si un fichier user.r n'est pas créé, et que REBOL ne peut trouver un fichier user.r existant dans son environnement, aucun paramétrage ne sera chargé. Voir le chapitre sur les Opérations pour plus d'informations.

Pour modifier les paramétrages réseau, saisissez **set-user** à l'invite de commande. Ceci relance le même script de configuration pour le réseau exécuté lorsque REBOL démarre pour la première fois. Ce script est chargé à partir du fichier rebol.r. Si ce fichier ne peut être trouvé, ou si vous voulez éditer le paramétrage directement, vous pouvez utiliser un éditeur de texte pour modifier le fichier user.r.

Au sein du fichier user.r, les paramétrages réseau se trouvent dans un bloc qui suit la fonction set-net. Au minimum, le bloc devrait contenir deux items :

- Votre adresse email à utiliser dans les champs "From" et "Reply" d'un email et pour un login anonyme.
- Votre serveur par défaut ; ce peut être également votre serveur de mail primaire.

De plus, vous pouvez indiquez un certain nombre d'autres items :

- Un serveur différent pour le mail entrant (pour POP).
- Un serveur proxy (pour se connecter au réseau).
- Un numéro de port pour le proxy.
- Le type de proxy (voir les paramétrages du Proxy ci-dessous)

Vous pouvez aussi ajouter des lignes après la fonction set-net pour configurer d'autres genres de protocoles. Par exemple, vous pouvez définir les valeurs de time-out pour les protocoles, définir un mode FTP passif, un identifier "user-agent" pour le HTTP, des proxys séparés pour des protocoles différents, et plus.

Un exemple de bloc **set-net** est :

```
set-net [user@domain.dom mail.server.dom]
```

Le premier champ spécifie votre adresse email, et le second champ indique votre serveur de mail par défaut (remarquez qu'il n'est pas nécessaire de mettre des guillemets ici). Pour la plupart des réseaux, c'est suffisant et aucun autre paramétrage n'est nécessaire (à moins que vous n'utilisiez un

serveur proxy). Votre serveur par défaut sera également utilisé si aucun autre serveur spécifique n'est mentionné.

De plus, si vous utilisez un serveur POP (pour les courriers entrants) différent de votre serveur SMTP (courrier sortant), vous pouvez le spécifier aussi

```
set-net [
    user@domain.dom
    mail.server.dom
    pop.server.dom
]
```

Toutefois, si les serveurs POP et SMTP sont les mêmes, ceci n'est pas nécessaire.

4.2 Paramétrages du Proxy

Si vous utilisez un proxy ou un pare-feu (firewall), vous pouvez fournir à la fonction **set-net** les paramétrages du proxy. Ceci comprend le nom ou l'adresse du serveur proxy, un numéro de port pour accéder au serveur, et en option, le type de proxy.

Par exemple :

```
set-net [
    email@addr
    mail.example.com
    pop.example.com
    proxy.example.com
    1080
    socks
]
```

Cet exemple utilisera un serveur de proxy appelé proxy.example.com sur son port TCP 1080 avec la méthode "socks" pour le proxy. Pour utiliser un serveur socks4, utilisez le mot "socks4" au lieu de socks. Pour utilisez le serveur générique CERN, utilisez le mot "generic".

Vous pouvez aussi définir un serveur de proxy spécifique pour un scheme (protocole).

Chaque protocole possède son propre objet proxy que vous pouvez adapter spécifiquement selon le scheme. Voici un exemple de paramétrages de proxy pour FTP :

```
system/schemes/ftp/proxy/host: "proxy2.example.com"

system/schemes/ftp/proxy/port-id: 1080

system/schemes/ftp/proxy/type: 'socks
```

Dans ce cas, seul le protocole FTP utilise un serveur de proxy spécial. Notez que chaque nom de machine doit être une chaîne et que le type de proxy doit être un mot littéral.

Voici deux exemples supplémentaires. Le premier exemple définit un proxy de type générique (CERN) pour le HTTP :

```
system/schemes/http/proxy/host: "wp.example.com"

system/schemes/http/proxy/port-id: 8080

system/schemes/http/proxy/type: 'generic
```

Dans l'exemple ci-dessus, toutes les requêtes HTTP passent à travers un proxy de type générique sur l'adresse wp.example.com en utilisant le port 8080.

```
system/schemes/smtp/proxy/host: false

system/schemes/smtp/proxy/port-id: false

system/schemes/smtp/proxy/type: false
```

Dans l'exemple ci-dessus, l'intégralité du courrier sortant ne passe pas par un serveur de proxy. La valeur **false** empêche que le serveur de proxy par défaut soit utilisé. Si vous mettiez ces champs à none, alors le serveur de proxy par défaut sera utilisé, s'il a été configuré.

Si vous voulez contourner (bypasser) les paramétrages de proxy pour des machines particulières, comme celles situées sur votre réseau local, vous pouvez fournir une liste de machines autorisées (bypass list).

Voici une liste pour le serveur de proxy par défaut :

```
system/schemes/default/proxy/bypass:
    ["host.example.net" "*.example.com"]
```

Notez que l'astérisque **(*)** et le point d'interrogation **(?)** peuvent être utilisés pour des correspondances de machines. L'astérisque **(*)** est utilisé dans l'exemple précédent pour autoriser toutes les machines dont le nom se termine par *example.com*.

Pour définir une liste de machines autorisées seulement pour le scheme HTTP, utilisez :

```
system/schemes/http/proxy/bypass:
    ["host.example.net" "*.example.com"]
```

4.3 Autres paramétrages

En supplément des paramètres du proxy, vous pouvez définir des valeurs de time-out pour tous les schemes (par défaut) ou pour des schemes spécifiques. Par exemple, pour augmenter la valeur du time-out pour tous les schemes, vous pouvez écrire :

```
system/schemes/default/timeout: 0:05
```

Ceci définit un time-out réseau de 5 minutes. Si vous voulez juste augmenter le time-out pour le scheme SMTP, vous pouvez écrire :

```
system/schemes/smtp/timeout: 0:10
```

Certains schemes possèdent des champs personnalisés. Par exemple, le scheme FTP vous permet de définir un mode passif pour tous les transferts :

```
system/schemes/ftp/passive: on
```

Le mode FTP passif est pratique car les serveurs FTP configurés ainsi n'essayent pas de se connecter en retour au travers de votre pare-feu.

Lorsque vous essayer d'accéder à des sites Web, vous pouvez vouloir utiliser un champ "user-agent" différent dans la requête HTTP, afin d'obtenir de meilleurs résultats sur les quelques sites qui détectent le type de navigateur :

system/schemes/http/user-agent : "Mozilla/4.0"

4.4 Accéder aux paramétrages

Chaque fois que REBOL démarre, il lit le fichier user.r pour trouver ses paramétrages réseau. Ces paramétrages sont réalisés avec la fonction **set-net**.

Les scripts peuvent accéder à ces paramètres au travers de l'objet system/schemes.

```
system/user/email ; used for email from and reply
system/schemes/default/host - your primary server
system/schemes/pop/host - your POP server
system/schemes/default/proxy/host - proxy server
system/schemes/default/proxy/port-id - proxy port
system/schemes/default/proxy/type - proxy type
```

Ci-dessous se trouve une fonction qui renvoie un bloc contenant le paramétrage réseau dans le même ordre que la fonction **set-net** les acceptent :

```
get-net: func [][
    reduce [
    system/user/email
    system/schemes/default/host
    system/schemes/pop/host
    system/schemes/default/proxy/host
    system/schemes/default/proxy/port-id
    system/schemes/default/proxy/type
    ]
]
```

```
probe get-net
```

5 DNS - Domain Name Service

DNS est le service réseau qui traduit les noms de domaine en leur adresse IP. De surcroît, vous pouvez utiliser DNS pour trouver une machine et son nom de domaine à partir d'une adresse IP.

Le protocole DNS peut être utilisé de trois manières : vous pouvez rechercher l'adresse IP primitive d'un nom de machine, ou un nom de domaine pour une adresse IP, et vous pouvez trouver le nom et l'adresse IP de votre machine locale.

Pour retrouver l'adresse IP d'une machine spécifique dans un domaine spécifique, saisissez :

```
print read dns://www.rebol.com
207.69.132.8
```

Vous pouvez aussi obtenir le nom de domaine qui est associé avec une adresse IP particulière :

```
print read dns://207.69.132.8
rebol.com
```

Notez qu'il n'est pas incongru pour cette recherche en reverse DNS de retourner le résultat **none**. Il existe des machines qui n'ont pas de noms.

```
print read dns://11.22.33.44
none
```

Pour déterminer le nom de votre système, essayez la lecture DNS d'une URL vide de la forme :

```
print read dns://
crackerjack
```

Les données renvoyées ici dépendent du type de la machine. Cela peut être un nom de machine sans domaine, comme indiqué précédemment, mais être aussi un nom de machine complet, comme crackerjack.example.com. Ceci dépend du système d'exploitation et de la configuration réseau du système.

Voici un exemple qui recherche et affiche les adresses IP pour un certain nombre de sites Web :

```
domains: [
    www.rebol.com
    www.rebol.org
    www.mochinet.com
    www.sirius.com
]

foreach domain domains [
    print ["address for" domain "is:"
```

```
    read join dns:// domain]
]
address for www.rebol.com is: 207.69.132.8
address for www.rebol.org is: 207.66.107.61
address for www.mochinet.com is: 216.127.92.70
address for www.sirius.com is: 205.134.224.1
```

6 Whois

le protocole whois renvoie des informations concernant des noms de domaines depuis un référentiel central. Le service Whois est fourni par les organisations qui font fonctionner Internet. Whois est souvent utilisé pour retrouver les informations d'enregistrement d'un domaine Internet ou d'un serveur.

Il peut vous dire qui est propriétaire du domaine, comment leur contact technique peut être joint, et d'autres informations.

Pour obtenir ces informations, utilisez la fonction **read** avec une URL Whois.

Cette URL doit contenir le nom de domaine et le nom du serveur Whois séparés par un signe **(@)**.

Par exemple, pour obtenir des informations concernant *example.com* depuis le référentiel Internet :

```
print read whois://example.com@rs.internic.net
connecting to: rs.internic.net
Whois Server Version 1.1
Domain names in the .com, .net, and .org domains can now be
registered with many different competing registrars. Go to
http://www.internic.net for detailed information.
Domain Name: EXAMPLE.COM
Registrar: NETWORK SOLUTIONS, INC.
Whois Server: whois.networksolutions.com
Referral URL: www.networksolutions.com
Name Server: NS.ISI.EDU
Name Server: VENERA.ISI.EDU
Updated Date: 17-aug-1999
>>
The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains
and Registrars.
```

Le code précédent est seulement un exemple. Le détail de l'information renvoyée, et les serveurs qui supportent Whois changent de temps en temps.

Si au lieu d'un nom de domaine, vous fournissez un mot, toutes les entrées qui correspondent à ce mot seront renvoyées :

```
print read whois://example@rs.internic.net
connecting to: rs.internic.net
```

```
Whois Server Version 1.1
Domain names in the .com, .net, and .org domains can now be
registered with many different competing registrars. Go to
http://www.internic.net for detailed information.
EXAMPLE.512BIT.ORG
EXAMPLE.ORG
EXAMPLE.NET
EXAMPLE.EDU
EXAMPLE.COM
To single out one record, look it up with "xxx", where xxx is one
of the of the records displayed above. If the records are the same, look them
up with "=xxx" to receive a full display for each record.
>>
The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains
and Registrars.
```

Le protocole Whois n'accepte pas les URLs, comme www.example.com, à moins que l'URL fasse partie du nom de la société enregistrée.

7 Finger

Le protocole finger retrouve des informations spécifiques à un utilisateur stockées dans le fichier log de l'utilisateur.

Pour pouvoir demander des informations sur un utilisateur à un serveur, celui-ci doit exécuter ce protocole finger. L'information est demandée en appelant avec **read** une URL finger qui comprend le nom de l'utilisateur et un nom de domaine, et se présente au format email :

```
print read finger://username@example.com
```

L'exemple précédent renvoie les informations concernant l'utilisateur référencé par username@example.com. L'information retournée dépend de celle que l'utilisateur a fourni et des paramétrages du serveur finger. Egalement, les détails de l'information retournée sont propres à chaque serveur ; les exemples ci-dessous décrivent seulement des serveurs génériques. La plupart des serveurs peuvent avoir des comportements non standards sur les requêtes finger.

Par exemple, l'information suivante pourrait être retournée :

```
Login: username
Name: Firstname Lastname
Directory: /home/user
Shell: /usr/local/bin/tcsh
Office: City, State +1 555 555 5555
Last login Wed Jul 28 01:10 (PDT) on ttyp0 from some.example.com
No Mail.
No Plan.
```

Remarquez que finger informe de la dernière connexion de l'utilisateur sur la machine, et aussi s'il y a

des courriers électroniques en attente pour lui. Si l'utilisateur lit un email à partir de son compte, parfois finger fournit cette information : lorsque l'email a été reçu et la dernière fois que l'utilisateur a récupéré son email :

```
New mail received Sun Sep 26 11:39 1999 (PDT)
Unread since Tue Sep 21 04:45 1999 (PDT)
```

Le serveur finger peut aussi renvoyer le contenu d'un fichier plan ou un fichier projet s'ils existent. Les utilisateurs peuvent inclure n'importe quelle information qu'ils souhaitent dans un fichier plan ou projet.

Il est aussi possible de retrouver des informations sur les utilisateurs en utilisant leur nom ou leur prénom. Les serveurs finger ont besoin que vous mettiez en majuscules

Certains serveurs finger demandent que soient écrits en majuscule les noms tels qu'ils apparaissent dans le fichier de login ou dans le fichier en ligne utilisé par le serveur finger, pour retrouver les informations sur l'utilisateur. D'autres serveurs finger sont plus tolérants vis-à-vis de la mise en majuscules.

Un serveur finger répondra aux requêtes sur le vrai nom en renvoyant toutes les listes qui correspondent aux critères de recherches. Par exemple, si il y a plusieurs utilisateurs qui possèdent le même prénom zaphod, si vous saisissez la requête :

```
print read finger://Zaphod@main.example.com
```

celle-ci renverra tous les utilisateurs ayant pour prénom ou nom zaphod. Certains serveurs finger renvoient un listing d'utilisateurs quand le nom de l'utilisateur est omis. Par exemple, le code :

```
print read finger://main.example.com
```

retournera une liste de tous les utilisateurs connectés sur la machine, si le service finger installé sur celle-ci l'autorise.

Certaines machines limitent le service finger pour des raisons de sécurité. Elles peuvent demander un nom d'utilisateur valide, et renvoyer seulement les informations liées à cet utilisateur. Si vous interrogez un serveur finger de ce genre, sans fournir des informations sur l'utilisateur, le serveur vous répondra qu'il attend des informations sur un utilisateur spécifique.

Si un serveur ne supporte pas le protocole finger, REBOL retourne une erreur d'accès :

```
print read finger://host.dom
connecting to: host.dom
Access Error: Cannot connect to host.dom.
Where: print read finger://host.dom
```

8 Daytime - Network Time Protocol

Le protocole daytime retourne le jour et l'heure courante. Pour se connecter à une serveur daytime,

utiliser **read** avec une URL daytime. Cette URL comprend le nom du serveur devant renvoyer la date :

```
print read daytime://everest.cclabs.missouri.edu
Fri Jun 30 16:40:46 2000
```

Le format de l'information renvoyée par les serveurs peut varier, selon le serveur. Notez que l'indication du fuseau horaire peut ne pas être présente.

Si le serveur que vous interrogez ne supporte pas le protocole daytime, REBOL renvoie une erreur :

```
print read daytime://www.example.com
connecting to: www.example.com
** Access Error: Cannot connect to www.example.com.
** Where: print read daytime://www.example.com
```

9 HTTP - Hyper Text Transfer Protocol

Le Web (World Wide Web - WWW) est caractérisé par deux technologies fondamentales : HTTP, et HTML. HTTP est l'acronyme de "Hyper Text Transfer Protocol", le protocole qui contrôle comment des serveurs Web et des navigateurs Web communiquent les uns avec les autres. HTML signifie "Hyper Text Markup Language" qui définit la structure et le contenu d'une page Web.

Pour récupérer une page Web, le navigateur envoie sa requête à un serveur Web utilisant HTTP. A la réception de la requête, le serveur l'interprète, parfois en utilisant des scripts CGI (voir CGI - Common Gateway Interface), et renvoie des données. Ces données peuvent être n'importe quoi, dont du HTML, du texte, des images, des programmes ou du son.

9.1 Lecture d'une page Web

Pour lire une page Web, utilisez la fonction **read** avec une URL HTTP.

Par exemple :

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

Ceci retourne une page Web pour www.rebol.com. Notez qu'une chaîne de caractères qui contient le code HTML (NdT : c'est-à-dire le code source de la page HTML), est renvoyée par la commande **read**. Aucune image ou graphique, ni aucune information n'est ramenée. Pour cela, il est nécessaire d'effectuer des opérations de lectures en complément. La page Web peut être affichée sous la forme de code HTML en utilisant **print**, elle peut être écrite dans un fichier avec la commande **write**, ou être envoyée dans un email avec la commande **send**.

```
print page
```

```
write %index.html page
```

```
send zaphod@example.com page
```

La page peut être manipulée de bien des façons, en utilisant diverses fonctions REBOL, comme **parse**, **find**, et **load**.

Par exemple, pour chercher au sein d'une page Web toutes les occurrences du mot REBOL, vous pouvez écrire :

```
parse read http://www.rebol.com [
    any [to "REBOL" copy line to newline (print line)]
]
```

9.2 Scripts sur des sites Web

Un serveur Web peut manipuler plus que des scripts HTML. Les serveurs Web sont tout à fait pratiques pour fournir également des scripts REBOL.

Vous pouvez charger des scripts REBOL directement depuis un serveur Web avec la commande **load** :

```
data: load http://www.rebol.com/data.r
```

Vous pouvez aussi évaluer des scripts directement depuis un serveur Web avec la commande **do** :

```
data: do http://www.rebol.com/code.r
```

Avertissement : Soyez prudent avec cet usage de **do**. Evaluer sans précaution des scripts sur des serveurs Internet ouverts peut provoquer des dégâts. Evaluez un script uniquement si vous êtes certain de la fiabilité de sa source, si vous avez contrôlé sa source ou que vous avez conservé vos paramétrages de sécurité REBOL à leur plus haut niveau.

De plus, les pages Web qui contiennent du HTML peuvent aussi contenir des scripts REBOL insérés dedans, et peuvent être exécuter avec :

```
data: do http://www.rebol.com/example.html
```

Pour savoir si un script existe dans une page avant de l'évaluer, utilisez la fonction **script ?** .

```
if page: script? http://www.rebol.com [do page]
```

La fonction **script ?** lit la page depuis le site Web et renvoie celle-ci à partir de la position de l'en-tête REBOL.

9.3 Chargement de pages avec balises

Les pages HTML et XML peuvent être rapidement converties en bloc REBOL avec la fonction **load/markup**. Cette fonction renvoie un bloc constitué de toutes les balises et les chaînes de caractères trouvées dans la page. Tous les espaces et les sauts de ligne sont conservés.

Pour filtrer une page en ôtant toutes les balises Web et juste imprimer le texte, saisissez :

```
tag-text: load/markup http://www.rebol.com
text: make string! 2000

foreach item tag-text [
    if string? item [append text item]
]

print text
```

Vous pouvez alors effectuer une recherche dans ce texte avec des modèles. Il contient tous les espaces et les sauts de ligne du fichier HTML original.

Voici un autre exemple qui contrôle tous les liens trouvés dans une page Web pour s'assurer de l'existence des pages à laquelles ces liens font référence :

```
REBOL []

page: http://www.rebol.com/developer.html
set [path target] split-path page
system/options/quiet: true      ; turn off connexion msgs
tag-text: load/markup page
links: make block! 100

foreach tag tag-text [  ; find all anchor href tags
    if tag? tag [
    if parse tag [
        "A" thru "HREF="
        [{"} copy link to {"} | copy link to ">"]
        to end
    ][
        append links link
    ]
    ]
]

print links

foreach link unique links [    ; try each link
    if all [
    link/1  #"#"
    any [flag: not find link ":"
         find/match link "http:"]
```

```
    ][
    link: either flag [path/:link][to-url link]
    prin [link "... "]
    print either error? try [read link]
        ["failed"]["OK"]
    ]
]
```

9.4 Autres Fonctions

Pour vérifier si une page Web existe, utilisez la fonction **exists ?**, laquelle renvoie **true** si la page existe.

```
if exists? http://www.rebol.com [
    print "page still there"
]
```

Note :

Habituellement, il est plus rapide dans la plupart des cas de juste lire la page, plutôt que de vérifier d'abord si elle existe. Par ailleurs, le script appelle deux fois le serveur et ceci peut être assez consommateur de temps.

Pour connaître la date de dernière modification d'une page Web, utilisez la fonction **modified ?** :

```
print modified? http://www.rebol.com/developer.html
```

Cependant, tous les serveurs Web ne fournissent pas cette information de date de modification. Typiquement, les pages générées dynamiquement ne renvoient pas de date de modification.

Une autre manière de déterminer si une page Web a changé est de l'interroger régulièrement et de la contrôler. Une façon pratique de vérifier si la page Web a changé est d'utiliser la fonction **checksum**.

Si la précédente valeur de checksum calculée diffère de la valeur courante, cela signifie que la page Web a été changée depuis le dernier contrôle. Voici un exemple qui utilise cette technique. Il vérifie une page toutes les huit heures.

```
forever [
    page: read http://www.rebol.com
    page-sum: checksum page
    if any [
    not exists? %page-sum
    page-sum      ][
    print ["Page changed" now]
```

```
    save %page-sum page-sum
    send luke@rebol.com page
    ]
    wait 8:00
]
```

Lorsque la page est modifiée, elle est envoyée par email à Luke.

9.5 Agir comme un Navigateur

Normalement, REBOL s'identifie lui-même vis-à-vis d'un serveur Web quand il lit une page. Cependant, certains serveurs sont programmés pour répondre uniquement à certains navigateurs. Si une requête à un serveur ne retourne pas la bonne page Web, vous pouvez modifier la requête pour la rendre identique à une venant d'un autre type de navigateur Web.

S'identifier comme étant un navigateur Web particulier est fait par de nombreux programmes, afin de permettre à des sites Web de répondre correctement. Cependant, cette pratique peut conduire à faire échouer l'utilisation normale après l'identification du navigateur.

Pour changer les requêtes HTTP et leur donner une ressemblance avec celles envoyées par Netscape 4.0, vous pouvez modifiez la valeur du user-agent au sein de l'agent (handler) HTTP :

```
system/options/http/user-agent: "Mozilla/4.0"
```

Modifier cette variable affecte toutes les requêtes HTTP qui suivent.

9.6 Envoi de requêtes CGI

Les requêtes HTTP CGI peuvent être émises de deux manières. Vous pouvez inclure les données de la requête dans l'URL ou bien, vous pouvez fournir les données de la requête au travers d'une opération d'envoi HTTP (POST).

Une requête avec une URL CGI utilise une URL normale. L'exemple ci-dessous envoie au script CGI test.r la valeur 10 pour sa variable *data*.

```
read http://www.example.com/cgi-bin/test.r?data=10
```

L'émission d'une requête CGI avec post nécessite que vous fournissiez les données CGI en tant que partie du raffinement **custom** de la fonction **read**. L'exemple ci-dessous montre comment est émise la requête CGI :

```
read/custom http://www.example.com/cgi-bin/test.r [
    post "data: 10"
]
```

Dans cet exemple, le raffinement ***/custom*** est utilisé pour fournir des informations supplémentaires

pour la lecture avec **read**. Le second argument est un bloc qui débute avec le mot *post* et continue avec la chaîne à envoyer.

La méthode "post" est pratique pour envoyer facilement du code REBOL et des données à un serveur Web en mode CGI. L'exemple suivant illustre ceci :

```
data: [sell 10 shares of "ACME" at $123.45]

read/custom http://www.example.com/cgi-bin/test.r reduce [
    `post mold data
]
```

La fonction **mold** produira une chaîne formatée pour REBOL prête à être émise vers le serveur Web.

10 SMTP - Simple Mail Transport Protocol

Le protocole SMTP (Simple Mail Transport Protocol) détermine les transferts de messages électroniques via Internet. Le SMTP définit les interactions entre les serveurs Internet qui contribuent à relayer les courriers depuis leur expéditeur jusqu'à leur destinataire.

10.1 Envoi d'Email

Un courrier électronique est envoyé avec le protocole SMTP en utilisant la fonction **send**. Cette fonction peut expédier un courrier électronique vers une ou plusieurs adresses emails.

Pour que la fonction **send** opère correctement, vos paramètres réseau doivent être définis. La fonction **send** nécessite que vous spécifiez une adresse email (champ From d'un email), et votre serveur d'email par défaut. Voir le début de ce chapitre.

La fonction **send** prend deux arguments : une adresse email et un message.

Par exemple :

```
send user@example.com "Hi from REBOL"
```

Le premier argument doit être un email ou un bloc d'adresses emails (block). Le second argument peut être de n'importe quel type de données (datatype).

```
send luke@rebol.com $1000.00

send luke@rebol.com 10:30:40

send luke@rebol.com bill@ms.dom

send luke@rebol.com [Today 9-Apr-99 10:30]
```

Chacun de ces simples messages emails peut être interprété côté receveur (avec REBOL) ou visualisé avec un client normal de messagerie électronique. Vous pouvez envoyer un fichier complet d'abord en le lisant, puis en le passant comme second argument à la fonction **send** :

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

Des données binaires, comme des images ou des programmes exécutables, peuvent aussi être envoyées :

```
send luke@rebol.com read/binary %rebol
```

Les données binaires sont encodées de façon à permettre leur transfert sous forme de texte. Pour expédier un message binaire auto-extractible, vous pouvez écrire :

```
send luke@rebol.com join "REBOL for the job" [
    newline "REBOL []" newline
    "write/binary %rebol decompress "
    compress read/binary %rebol
]
```

Lorsque le message est réceptionné, le fichier peut être extrait en utilisant la fonction **do**.

10.2 Destinataires multiples

Pour envoyer un message à de multiples destinataires, vous pouvez utiliser un bloc d'adresses emails :

```
send [luke@rebol.com ben@example.com] message
```

Dans ce cas, chaque message est individuellement adressé avec seulement un nom de destinataire apparaissant dans le champ To (identique à l'adressage en copie cachée BCC).

Le bloc d'adresses email peut être de n'importe quelle longueur ou même être un fichier que vous chargez. Il vous faut juste être attentif à avoir des adresses emails valides, et non des chaînes de caractères qui, elles, sont ignorées.

```
friends: [
    bob@cnn.dom
    betty@cnet.dom
    kirby@hooya.dom
    belle@apple.dom
    ...
]
send friends read %newsletter.txt
```

10.3 Courrier en masse

Si vous expédiez du courrier électronique à un groupe important, vous pouvez réduire la charge sur votre serveur en distribuant à chacun dans le groupe un simple message. C'est l'objet du raffinement **/only**. Il utilise une propriété du protocole SMTP pour envoyer seulement un message à des adresses emails multiples. En utilisant la liste "friends" de l'exemple précédent :

```
send/only friends message
```

Les messages ne sont pas adressés individuellement. Vous pouvez avoir vu ce mode dans certains des emails que vous pouvez recevoir. Lorsque vous recevez un courrier en masse, votre adresse n'apparaît pas dans le champ To. Le mode d'envoi en masse du SMTP devrait être utilisé pour les listes de diffusion, et pas pour de l'envoi de Spam. Le Spam est contraire à la Net-étiquette, il est illégal dans de nombreux pays et états, et peut conduire à votre exclusion de votre Fournisseur d'Accès Internet, et d'autres sites.

10.4 Ligne de sujet et en-têtes

Par défaut, la fonction **send** utilise la première ligne de l'argument *message* comme ligne de sujet pour le courrier électronique. Pour fournir une ligne de sujet personnalisée, vous devrez donner un en-tête d'email à la fonction **send**.

En complément du sujet, vous pouvez indiquer une organisation, une date, un champ CC, et même vos propres champs personnalisés.

Pour indiquer un en-tête, utiliser le raffinement **/header** de la fonction **send**, et incluez l'en-tête sous forme d'un objet. L'objet servant d'en-tête doit être composé à partir de l'objet **system/standard/email**. Par exemple :

header : make system/standard/email [

```
Subject: "Seen REBOL yet?"
Organization: "Freedom Fighters"
```

]

Notez que les champs standards comme l'adresse From, ne sont pas requis et sont automatiquement complétés par la fonction send.

L'en-tête est ensuite fourni en tant qu'argument à **send/header** :

```
send/header friends message header
```

Le courrier électronique ci-dessus est émis en utilisant l'en-tête personnalisé pour chacun des messages.

10.5 Déboguer vos scripts

Lors des tests de vos scripts utilisant **send**, il est judicieux de vous expédier à vous même le courrier électronique d'abord, avant de l'expédier à d'autres. Vérifiez et testez scrupuleusement vos scripts pour être sûrs de ce que vous voulez réaliser. Une erreur commune est d'envoyer un nom de fichier plutôt que son contenu. Par exemple, si vous écrivez :

```
send person %the-data-file.txt
```

ceci envoie le nom du fichier, et non son contenu.

11 POP - Post Office Protocol

Le protocole POP (Post Office Protocol) vous permet de récupérer le courrier électronique qui attend dans votre boîte aux lettres, sur un serveur de mails. POP définit un certain nombre d'opérations sur la façon d'accéder à votre boîte aux lettres (BAL) et de stocker des emails sur votre serveur.

11.1 Lecture d'Email

Vous pouvez lire tout votre courrier électronique en une seule ligne sans effacer quoique ce soit de votre serveur de courrier. Ceci est réalisé en lisant avec POP une URL comprenant votre nom d'utilisateur (compte de courrier), votre mot de passe, et le serveur de mails.

```
mail: read pop://user:pass@mail.example.com
```

Les courriers sont renvoyés sous la forme d'un bloc de plusieurs chaînes de caractères, que vous pouvez afficher une par une avec un code comme celui-ci :

```
foreach message mail [print message]
```

Pour lire individuellement des emails depuis le serveur, vous aurez besoin d'ouvrir un port de connexion avec le serveur puis de gérer chaque message un par un. Pour ouvrir un port POP :

```
mailbox: open pop://user:pass@mail.example.com
```

Dans cet exemple, "mailbox" est traitée comme une série, et la plupart des fonctions standards propres aux séries sont utilisables comme length ?, first, second, third, pick, next, back, head, tail, head ?, tail ?, remove, et clear.

Pour déterminer le nombre de messages électroniques sur le serveur, utilisez la fonction length ?.

```
print length? mailbox
37
```

De plus, vous pouvez extraire la taille totale de tous les messages et leur taille individuellement avec :

```
print mailbox/locals/total-size
```

```
print mailbox/locals/sizes
```

NdT : on utilise ici une méthode de l'objet *mailbox* renvoyé par la fonction **open**.

Pour afficher le premier, le second, et le dernier message électronique, vous pouvez écrire :

```
print first mailbox

print second mailbox

print last mailbox
```

Vous pouvez aussi utiliser le fonction **pick** pour rapatrier un message spécifique :

```
print pick mailbox 27
```

Vous pouvez récuperer et afficher chaque message du plus ancien au plus récent en utilisant une boucle **loop** qui est identique à celle utilisée pour d'autres types de série :

```
while [not tail? mailbox] [
    print first mailbox
    mailbox: next mailbox
]
```

Vous aussi lire vos courriers électroniques du plus récent au plus ancien avec la boucle suivante :

```
mailbox: tail mailbox

while [not head? mailbox] [
    mailbox: back mailbox
    print first mailbox
]
```

Une fois terminées les opérations sur le port, fermez-le. Ceci est fait avec la ligne suivante :

```
close mailbox
```

11.2 Suppression d'emails

Comme avec les séries, la fonction **remove** peut être appelée pour effacer un seul message, et la fonction **clear** peut être utilisée pour effacer tous les messages depuis la position courante dans la liste, la série, jusqu'à la fin de *mailbox*.

Par exemple, pour lire un message, sauvez-le dans un fichier et effacez-le du serveur.

```
mailbox: open pop://user:pass@mail.example.com
write %mail.txt first mailbox
remove mailbox
close mailbox
```

Le message électronique est effacé du serveur lorsque la fonction **close** est exécutée. Pour effacer le 22ème message du serveur, vous pouvez écrire :

```
user:pass@mail.example.com
remove at mailbox 22
close mailbox
```

Vous pouvez effacer un nombre donné de messages en utilisant le raffinement **/part** avec la fonction **remove** :

```
remove/part mailbox 5
```

Pour effacer tous les messages de votre boîte aux lettres, utilisez la fonction **clear** :

```
mailbox: open pop://user:pass@example.com
clear mailbox
close mailbox
```

La fonction **clear** peut aussi est utilisée à différentes positions dans la série **mailbox**, de façon à n'effacer que les messages entre ces positions et jusqu'à la fin de la série.

11.3 Manipulation d'en-tête de courrier électronique

Les courriers électroniques peuvent inclure un en-tête. L'en-tête contient des informations sur l'expéditeur, le sujet, la date et d'autres champs.

En REBOL, les en-têtes d'email sont manipulés en tant qu'objets qui contiennent tous les champs nécessaires. Pour transformer un message email en objet, vous pouvez utiliser la fonction **import-email**. Par exemple :

```
msg: import-email first mailbox

print first msg/from  ; the email address
print msg/date
print msg/subject
print msg/content
```

Vous pouvez alors facilement écrire un filtre qui scanne votre courrier électronique pour les messages qui débutent par un sujet particulier :

```
mailbox: open pop://user:pass@example.com
```

```
while [not tail? mailbox] [
    msg: import-email first mailbox
    if find/match msg/subject "[REBOL]" [
    print msg/subject
    ]
    mailbox: next mailbox
]

close mailbox
```

Voici un autre exemple qui vous alerte lorsque un email provenant d'un groupe d'amis est reçu :

```
friends: [orson@rebol.com hans@rebol.com]

messages: read pop://user:pass@example.com

foreach message messages [
    msg: import-email message
    if find friends first msg/from [
    print [msg/from newline msg/content]
    send first msg/from "Got your email!"
    ]
]
```

Ce filtre de spam efface du serveur tous les messages qui ne contiennent pas votre adresse email quelque part dans le message :

```
mailbox: open pop://user:pass@example.com

while [not tail? mailbox] [
    mailbox: either find first mailbox user@example.com
    [next mailbox][remove mailbox]
]

close mailbox
```

Voici une simple liste email qui reçoit des messages et les envoie à un groupe. Le serveur accepte juste les courriers des personnes du groupe.

```
group: [orson@rebol.com hans@rebol.com]

mailbox: open pop://user:pass@example.com

while [not tail? mailbox] [
    message: import-email first mailbox
    mailbox: either find group first message/from [
    send/only group first mailbox
    remove mailbox
    ][next mailbox]
```

```
]

close mailbox
```

12 FTP - File Transfer Protocol

Le protocole FTP (File Transfer Protocol) est extrêmement utilisé sur Internet pour transférer des fichiers depuis et vers une machine distante. Le FTP est couramment utilisé pour télécharger et mettre à jour des pages d'un site Web, et pour avoir en ligne des archives de fichier (sites de téléchargement).

12.1 Utilisation de FTP

Avec REBOL, les opérations relatives au protocole FTP sont effectuées de la même manière que si on avait des fichiers locaux.

Les fonctions telles que **read**, **write**, **load**, **save**, **do**, **open**, **close**, **exists ?**, **size ?**, **modified ?**, et d'autres encore sont utilisables avec FTP.

REBOL fait la distinction entre les fichiers locaux et les fichiers accessibles par FTP, au moyen de l'utilisation d'une URL FTP.

L'accès à des serveurs FTP peut être libre ou contrôlé. Des accès libres permettent à n'importe qui de se connecter au site FTP et de télécharger des archives, des fichiers. Ceci s'appelle un accès anonyme et est fréquemment utilisé pour des sites de téléchargement publics.

Les accès contrôlés nécessitent que vous fournissiez un nom d'utilisateur et un mot de passe pour accéder au site. C'est le principe pour la mise à jour de pages Web sur un site Web.

Bien que le protocole FTP ne requiert pas que votre configuration réseau REBOL soit OK, si vous utilisez un accès anonyme, une adresse email est souvent demandée. Cette adresse est trouvée dans l'objet **system/user/email**.

Normalement, lorsque vous démarrez REBOL, cette information est définie à partir de votre fichier **user.r**. Voir le chapitre sur le premier démarrage pour plus de détails. Si vous utilisez le protocole FTP au travers d'un pare-feu ou d'un serveur proxy, FTP doit être configuré pour opérer en mode passif. Le mode passif ne nécessite pas des connexions en retour depuis le serveur FTP vers le client, pour des transferts de données. Ce mode crée seulement des connexions sortantes depuis votre machine et permet d'avoir un haut niveau de sécurité. Pour engager le mode passif, vous devez positionner une variable dans l'agent (handler) du protocole FTP.

```
system/schemes/ftp/passive: true
```

Si vous ignorez si ce mode est nécessaire, essayez d'abord sans. Si cela ne fonctionne pas, paramétrez la variable comme ci-dessus.

12.2 URLs FTP

A la base, une URL FTP possède la forme suivante :

```
ftp://user:pass@host/directory/file
```

Pour des accès anonymes, le nom d'utilisateur (user) et le mot de passe (password) peuvent être omis :

```
ftp://host/directory/file
```

La plupart des exemples dans cette section utilise cette forme simple ; cependant, ils marchent aussi avec un nom d'utilisateur et un mot de passe.

Pour atteindre un répertoire distant, terminez l'URL par le symbole "slash" (/), comme avec :

```
ftp://user:pass@host/directory/
```

```
ftp://host/directory/
```

```
ftp://host/
```

Vous trouverez plus loin plus d'informations sur l'accès à des répertoires distants.

Il est commode de placer l'URL dans une variable et d'utiliser les paths pour fournir des noms de fichiers. Ceci permet de faire référence à l'URL avec juste un mot.

Par exemple :

```
site: ftp://ftp.rebol.com/pub/
read site/readme.txt
```

Cette technique est mise en oeuvre dans les sections qui suivent.

12.3 Transfert de fichiers Texte

Le protocole FTP établit une distinction entre les fichiers texte et les fichiers binaires. Lors du transfert de fichiers texte, FTP convertit les caractères de fin de ligne. Cela n'est pas souhaitable pour les fichiers binaires.

Pour lire un fichier texte, passez à la fonction **read** une URL FTP :

```
file: read ftp://ftp.site.com/file.r
```

Ceci met le contenu du fichier dans une chaîne (ici, *file*). Pour écrire ce fichier localement, utilisez cette ligne :

```
write %file.r read ftp://ftp.site.com/file.r
```

La plupart des raffinements de la fonction **read** sont également utilisables. Par exemple, vous pouvez utilisez **read/lines** avec :

```
data: read/lines ftp://ftp.site.com/file.r
```

Cet exemple renvoie le fichier sous la forme d'un bloc de lignes. Voir le chapitre sur les Fichiers pour plus d'informations sur les raffinements de la fonction **read**.

Pour écrire un fichier texte sur le serveur FTP, utilisez la fonction **write** :

```
write ftp://ftp.site.com/file.r read %file.r
```

La fonction **write** peut prendre aussi des raffinements. Voir le chapitre sur les Fichiers.

Comme normalement avec les transferts de fichiers texte, toutes les fins de lignes seront correctement converties durant le transfert FTP.

Voici un simple script qui met à jour les fichiers de votre site Web :

```
site: ftp://wwwuser:secret@www.site.dom/pages

files: [%index.html %home.html %info.html]

foreach file files [write site/:file read file]
```

Ceci ne devrait pas être utilisé pour transférer des images ou des fichiers de son, qui sont binaires. Utilisez la technique montrée dans la section suivante sur le Transfert de fichiers binaires.

En complément des fonctions **read** et **write**, vous pouvez aussi utilisez **load**, **save**, et **do** avec FTP.

```
data: load ftp://ftp.site.com/database.r

save ftp://ftp.site.com/data.r data-block

do ftp://ftp.site.com/scripts/test.r
```

12.4 Transfert de fichiers binaires

Pour évitez la conversion des caractères de fin de ligne, lors du transfert de fichiers binaires (images, archives zippés, fichiers exécutables), utilisez le raffinement **/binary**.

Par exemple, pour lire un fichier binaire depuis un serveur FTP :

```
data: read/binary ftp://ftp.site.com/file
```

Pour faire en local une copie du fichier :

```
write/binary %file read/binary ftp://ftp.site.com/file
```

Pour écrire un fichier binaire sur un serveur :

```
write/binary ftp://ftp.site.com/file read/binary %file
```

Aucune conversion de fin de ligne n'est réalisée.

Pour transférer un ensemble de fichiers graphiques sur un site Web, utilisez le script :

```
site: ftp://user:pass@ftp.site.com/www/graphics

files: [%icon.gif %logo.gif %photo.jpg]

foreach file files [
    write/binary site/:file read/binary file
]
```

12.5 Ajout à des fichiers

Le protocole FTP vous permet aussi d'ajouter du texte ou des données à un fichier existant. Pour faire cela, utilisez le raffinement **write/append** comme cela est décrit dans le chapitre sur les Fichiers.

```
write/append ftp://ftp.site.com/pub/log.txt reform
    ["Log entry date:" now newline]
```

Ceci peut aussi être fait avec des fichiers binaires.

```
write/binary/append ftp://ftp.site.com/pub/log.txt
    read/binary %datafile
```

12.6 Consultation de répertoires

Pour lire le contenu d'un répertoire FTP distant, faites suivre le nom du répertoire d'un symbole "/" (slash).

```
print read ftp://ftp.site.com/
pub-files: read ftp://ftp.site.com/pub/
```

Le slash terminal (/) indique qu'il s'agit d'un accès à un répertoire et non à un fichier. Le slash n'est pas toujours nécessaire mais il est recommandé dés lors que vous savez que vous accédez à un répertoire.

Le bloc de noms de fichiers qui est renvoyé comprend tous les éléments du répertoire. Au sein du bloc, les noms de répertoires sont signalés avec un slash à la fin de leur nom.

Par exemple :

```
foreach file read ftp://ftp.site.com/pub/ [
    print file
]
readme.txt
rebol.r
rebol.exe
library/docs/
```

Vous pouvez aussi utilisez la fonction **dir ?** sur un élément pour déterminer s'il s'agit d'un fichier ou d'un répertoire.

12.7 Information concernant les fichiers

Les mêmes fonctions qui fournissent de l'information concernant les fichiers locaux peuvent aussi fournir des informations sur les fichiers distants FTP. Ceci inclut les fonctions **modified ?**, **size ?**, **exists ?**, **dir ?**, et **info ?**.

Vous pouvez utilisez la fonction exists ? pour savoir si un fichier existe :

```
if exists? ftp://ftp.site.com/pub/log.txt [
    print "Log file is there"
]
```

Ceci marche également avec les répertoires, mais pensez à inclure le slash final après le nom du répertoire :

```
if exists? ftp://ftp.site.com/pub/rebol/ [
    print read ftp://ftp.site.com/pub/rebol/
]
```

Pour connaître la taille ou la date de modification d'un fichier :

```
print size? ftp://ftp.site.com/pub/log.txt

print modified? ftp://ftp.site.com/pub/log.txt
```

Pour déterminer si un nom d'élément est celui d'un répertoire :

```
if dir? ftp://ftp.site.com/pub/text [
    print "It's a directory"
]
```

Vous pouvez obtenir toutes ces informations en une seule requête avec la fonction **info ?** :

```
file-info: info? ftp://ftp.site.com/pub/log.txt
```

```
probe file-info

print file-info/size
```

Pour effectuer la même action sur un répertoire :

```
probe info? ftp://ftp.site.com/pub/
```

Pour afficher le contenu d'un répertoire :

```
files: open ftp://ftp.site.com/pub/

forall files [
    file: first files
    info: info? file
    print [file info/date info/size info/type]
]
```

12.8 Créer un répertoire

De nouveaux répertoires FTP peuvent être créés avec la fonction **make-dir** :

```
make-dir ftp://user:pass@ftp.site.com/newdir/
```

12.9 Suppression de fichiers

En supposant que vous ayez les permissions appropriées pour cela, des fichiers peuvent être supprimés du serveur FTP en utilisant la fonction **delete** :

```
delete ftp://user:pass@ftp.site.com/upload.txt
```

Vous pouvez aussi effacer des répertoires :

```
delete ftp://user:pass@ftp.site.com/newdir/
```

Notez que le répertoire doit être vide pour que sa suppression puisse se faire.

12.10 Renommage de fichiers

Vous pouvez renommer un fichier avec la ligne :

```
rename ftp://user:pass@ftp.site.com/foo.r %bar.r
```

Le nouveau nom du fichier sera bar.r.

Le protocole FTP permet aussi de déplacer un fichier vers un autre répertoire avec :

```
rename ftp://user:pass@ftp.site.com/foo.r %pub/bar.r
```

Pour renommer un répertoire sur un site FTP, là encore n'oubliez pas le slash à la fin du nom du répertoire :

```
rename ftp://user:pass@ftp.site.com/rebol/ rebol-old/
```

12.11 Au sujet des mots de passe

Les exemples ci-dessus incluent le mot de passe au sein des URLs, mais si vous prévoyez de partager votre script, vous ne voulez probablement pas que cette information soit connue. Voici une manière simple de demander le mot de passe via un interrogation en ligne de commande (prompt) et de construire l'URL adéquate :

```
pass: ask "Password? "

data: read join ftp://user: [pass "@ftp.site.com/file"]
```

Ou, vous pouvez demander à la fois le nom de l'utilisateur et le mot de passe :

```
user: ask "Username? "
pass: ask "Password? "
data: read join ftp:// [
    user ":" pass "@ftp.site.com/file"
]
```

Vous pouvez aussi ouvrir une connexion FTP en spécifiant un port plutôt qu'une URL. Ceci vous permet d'utiliser n'importe quel mot de passe, même ceux pouvant contenir des caractères spéciaux qui ne sont pas facile à écrire dans une URL.

Un exemple de spécification pour un port ouvrant une connexion FTP serait :

```
ftp-port: open [
    scheme: `ftp
    host: "ftp.site.com"
    user: ask "Username? "
    pass: ask "Password? "
]
```

Voir la partie sur la spécification de ressources réseau ci-dessus pour plus de détail.

12.12 Transfert de fichiers volumineux

Le transfert de fichiers volumineux nécessite quelques considérations particulières. Vous souhaitez sans doute transférer un fichier par morceaux pour réduire la quantité de mémoire requise par votre ordinateur, et pour fournir à l'utilisateur un retour sur l'évolution du transfert.

Voici un exemple qui télécharge un très gros fichier par morceaux.

```
inp: open/binary/direct ftp://ftp.site.com/big-file.bmp
out: open/binary/new/direct %big-file.bmp
buf-size: 200000
buffer: make binary! buf-size + 2

while [not zero? size: read-io inp buffer buf-size][
    write-io out buffer size
    total: total + size
    print ["transferred:" total]
]
```

Utilisez absolument le raffinement **/direct**, faute de quoi le fichier entier sera mis en buffer en interne de REBOL. Les fonctions **read-io** et **write-io** permettent de réutiliser la mémoire du buffer qui a déjà été allouée. D'autres fonctions comme **copy** alloue de la mémoire supplémentaire.

Si le transfert s'interrompt, vous pouvez redémarrer le transfert FTP à partir de l'endroit où il s'est arrêté. Pour cela, examinez le fichier résultant (out) ou la taille pour déterminer d'où recommencer le transfert.

Ouvrez à nouveau le fichier avec le raffinement **/custom** en spécifiant le mot restart et l'endroit d'où redémarrer la lecture.

Voici un exemple avec la fonction **open** où la variable "total" indique la longueur déjà lue du fichier :

```
inp: open/binary/direct/custom
ftp://ftp.site.com/big-file.bmp
reduce ['restart total]
```

Notez que le redémarrage d'un transfert FTP fonctionne seulement avec des transferts binaires. Il ne peut être effectué avec des transferts de fichiers texte parce que les conversions de caractères de fin de ligne induisent des modifications de taille.

13 NNTP - Network News Transfer Protocol

Le protocole NNTP (Network News Transfer Protocol) est la base pour des dizaines de milliers de newsgroups qui assurent un forum public pour des millions d'utilisateurs d'Internet. REBOL comprend deux niveaux de support pour le protocole NNTP.

- Le support interne qui autorise des fonctionnalités et des accès très limités. C'est le *scheme NNTP*.
- Un niveau supérieur de fonctionnalité qui est assuré par le scheme **news**, implémenté dans le fichier appelé **nntp.r**.

13.1 Lecture d'une liste de newsgroup

NNTP comprend deux composants : une liste de newsgroups supportés par un serveur de newsgroup dédié (typiquement, les newsgroups sont sélectionnés par les fournisseurs d'accès à Internet) ; et une base de données de messages en cours qui se rapportent à des newsgroups particuliers.

Pour retrouver la liste des messages de tous les newsgroups pour un serveur de news spécifique, utilisez la fonction **read** avec une URL NNTP telle que :

```
groups: read nntp://news.example.com
```

Cette opération peut durer un certain temps, selon votre connexion ; il y a des milliers de newsgroups.

13.2 Lire tous les messages

Si vous utilisez une connexion rapide, vous pouvez lire tous les messages relatif à un newsgroup avec :

```
messages: read nntp://news.example.com/alt.test
```

Cependant, soyez prudents. Certains newsgroups peuvent avoir des milliers de messages. Cela peut prendre un long moment pour télécharger tous les messages, et être très consommateur en mémoire, pour les manipuler.

13.3 Lecture de messages particuliers

Pour lire des messages spécifiques, ouvrez NNTP avec un port, et utilisez les fonctions relatives aux séries pour accéder aux messages. C'est assez similaire au fonctionnement vu précédemment pour la lecture de vos emails avec un port POP.

Par exemple :

```
group: open nntp://news.example.com/alt.test
```

Vous pouvez utilisez la fonction **length ?** pour déterminer le nombre de messages valables pour le newsgroup :

```
print length? group
```

Pour lire le premier message pour le newsgroup, utilisez la fonction **first** :

```
message: first group
```

Pour sélectionner un message spécifique dans le groupe, via son index, utilisez **pick** :

```
message: pick group 37
```

Pour créer un simple boucle permettant de scanner tous les messages contenant un mot-clé :

```
forall group [
    if find msg: first first group "REBOL" [
    print msg
    ]
]
```

Rappelez-vous qu'à la fin de la boucle, la série est positionnée sur sa fin (tail). Si vous avez besoin de revenir au début de la série des messages :

```
group: head group
```

N'oubliez pas non plus de fermer le port une fois que vous avez terminé de l'utiliser :

```
close group
```

13.4 Manipulation des en-têtes de News

Les messages des news incluent systématiquement un en-tête. L'en-tête stocke des informations sur l'expéditeur, le résumé, des mot-clés, le sujet, la date, et d'autres champs également.

Les en-têtes sont manipulés sous la forme d'objet REBOL. Pour convertir un message de news, en objet d'en-tête, vous pouvez utilisez la fonction **import-email**.

Par exemple,

```
message: first first group
header: import-email message
```

Vous pouvez à présent accéder aux différents champs du message de news :

```
print [header/from header/subject header/date]
```

Les différents newsgroups et les différents clients utilisent différents champs pour leurs en-têtes. Pour voir les champs valables pour un message particulier, affichez le premier item de l'objet d'en-tête, ici appelé header :

```
print first header
```

13.5 Expédier un message

Avant d'envoyer un message, vous devez créer un en-tête pour lui. Voici un en-tête générique qui peut être utilisé pour les newsgroups :

```
news-header: make object! [
    Path: "not-for-mail"
    Sender: Reply-to: From: system/user/email
    Subject: "Test message"
    Newsgroups: "alt.test"
    Message-ID: none
    Organization: "Docs For All"
    Keywords: "Test"
    Summary: "A test message"
]
```

Avant de l'envoyer, vous devez créer un numéro d'identification global pour lui. Voici une fonction qui réalise cela :

```
make-id: does [
    rejoin [
    "     system/user/email/user
    "."
    checksum form now
    "."
    random 999999
    "@"
    read dns://
    ">"
    ]
]
print news-header/message-id: make-id
```

A présent, vous pouvez combiner l'en-tête avec le message. Ils doivent être séparés par au moins une ligne blanche. Le contenu du message est lu à partir d'un fichier.

```
write nntp://news.example.net/alt.test rejoin [
    net-utils/export news-header
    newline newline
    read %message.txt
    newline
]
```

14 CGI - Common Gateway Interface

Le mode CGI (Common Gateway Interface) est utilisé avec de nombreux serveurs Web pour effectuer des traitements en plus et au delà de l'interface Web normale.

Les requêtes CGI sont soumises par des navigateurs Web à des serveurs Web. Typiquement, lorsque qu'un serveur reçoit une requête CGI, il exécute un script qui traite la requête et renvoie un résultat au navigateur. Ces scripts CGI sont écrits dans de très nombreux langages, et l'un des manières les faciles de manipuler du CGI est d'utiliser REBOL.

14.1 Paramétrage du serveur CGI

Le paramètrage d'un accés CGI est différent pour chaque serveur Web. Voir les instructions fournies avec votre serveur.

Typiquement, un serveur possède une option permettant d'activer le mode CGI. Vous devez activer cette option et fournir un chemin vers le répertoire où se trouvent vos scripts CGI. Un répertoire courant pour les scripts CGI s'appelle *cgi-bin*.

Sur les serveurs Web Apache, l'option ExecCGI active le mode CGI, et vous devez indiquer un répertoire (cgi-bin) pour vos scripts. C'est le mode de fonctionnement par défaut d'Apache.

Pour configurer CGI pour Microsoft IIS, allez dans les propriétés pour cgi-bin, et cliquez sur le bouton pour la configuration. Sur le panneau de configuration, cliquez sur "add" et entrez le chemin vers l'exécutable rebol.exe. La syntaxe pour cela est :

```
C:ebolebol.exe -cs %s %s
```

Les deux symboles %s sont nécessaires pour passer correctement le script et les arguments en ligne de commande à REBOL. Ajoutez l'extension propre aux fichiers REBOL (.r), et mettez le dernier champ sur PUT, DELETE. L'item "script engine" n'a pas besoin d'être sélectionné.

L'option -cs qui est passée à REBOL permet le mode CGI et autorise le script à accéder à tous les fichiers (!! Voir notes ci-dessous sur comment les scripts peuvent limiter les accès à des fichiers pour des répertoires spécifiques.)

D'autres serveurs Web que ceux décrits au-dessus nécessitent d'être configuré pour pouvoir exécuter l'exécutable REBOL avec des fichiers portant l'extension .r et avec l'option demandée -cs.

14.2 Scripts CGI

Avant de pouvoir exécuter un script sur la plupart des serveurs CGI, celui-ci devra disposer des permissions de fichiers adéquates. Sur les systèmes de type Unix ou ceux utilisant un serveur Apache, il sera nécessaire de modifier les permissions pour autoriser le script en lecture et en exécution pour tous les utilisateurs. Cela peut être fait avec la fonction Unix chmod.

Si vous être débutant dans ces concepts, vous devriez lire le manuel de votre système d'exploitation, ou demander à votre administrateur système avant de modifier les permissions du fichier.

Pour Apache, et divers autres serveurs Web qui exécutent des scripts REBOL, vous devez placer un en-tête dédié au début de chaque script. L'en-tête indique le chemin vers l'exécutable REBOL et l'option -cs. Voici un simple script CGI qui affiche la chaîne de caractères "hello !".

```
#!/path/to/rebol -cs
```

```
REBOL [Title: "CGI Test Script"]

print "Content-Type: text/plain"

print ""  ; required

print "Hello!"
```

De nombreuses choses peuvent empêcher un script CGI de fonctionner correctement. Testez d'abord ce simple script avant d'en essayez de plus complexe. Si votre script ne marche pas, voici quelques points à vérifiez :

- Vous avez activé l'option CGI sur votre serveur Web.
- La première ligne du script commence par # ! et le chemin exact vers REBOL.
- L'option -cs est fournie à REBOL.
- Le script commence avec l'affichage de "Content-Type :" (!!voir ci-dessous)
- Le script est dans le bon répertoire. (normalement, le répertoire cgi-bin).
- Le script a les permissions adéquates (lecture et exécution pour tous).
- Le script contient le saut de ligne nécessaire. Certains serveurs n'exécuteront pas le script s'il celui-ci ne contient pas le caractère CR (carriage return) pour les sauts de lignes. Vous devrez convertir le fichier. (Utilisez REBOL pour faire cela en une ligne : write file read file).
- Le script ne contient pas d'erreurs. Testez-le hors du mode CGI pour être sûr que le script se charge (n'a pas d'erreurs de syntaxe) et fonctionne proprement. Fournissez lui quelques données en exemple, et testez-le avec.
- Tous les fichiers auquel le script doit accéder ont les permissions adéquates.

Souvent l'un ou plusieurs de ces points n'est pas correct et empêche votre script de marcher. Vous aurez une erreur au lieu de voir une page Web. Si cette erreur est du type "Server Error" ou "CGI error", alors typiquement, il y a quelque chose à faire avec les droits ou le paramétrage du script. S'il s'agit d'un message d'erreur REBOL, alors le script est exécuté, mais vous avez une erreur à l'intérieur du script.

Dans le script présenté ci-dessus en exemple, la ligne "Content-Type" est critique. C'est la partie de l'en-tête HTTP qui est renvoyée vers le navigateur et qui l'informe du type de contenu qu'il va recevoir. Cette ligne est suivie d'une ligne vierge, qui la sépare du contenu.

Différents types de contenu peuvent être retourné. L'exemple précédent était en texte, mais vous pouvez aussi retourner du HTML comme montré dans l'exemple suivant. (Voir le manuel de votre serveur Web pour plus d'informations sur les types de contenu).

Le type de contenu et la ligne vierge peuvent être combinées en une seule ligne. L'ajout du symbole (^/) (accent circonflexe suivi d'un slash) est assimilable à une ligne vierge, qui effectue la séparation d'avec le contenu.

```
print "Content-Type: text/plain^/"
```

C'est une bonne habitude de toujours afficher cette ligne immédiatement au début de votre script. Cela permet le renvoi des messages d'erreur vers le browser si votre script rencontre une erreur.

Voici un simple script CGI qui affiche l'heure :

```
#!/path/to/rebol -cs

REBOL [Title: "Time Script"]

print "Content-Type: text/plain^/"

print ["The time is now" now/time]
```

14.3 Générer du contenu HTML

Il y a autant de manières de créer du contenu HTML qu'il y a de façons de créer des chaînes de caractères. Ce code génère une page qui affiche un compteur du nombre de visiteurs :

```
#!/path/to/rebol -cs

REBOL [Title: "HTML Example"]

print "Content-Type: text/html^/"

count: either exists? %counter [load %counter][0]
save %counter count: count + 1

print [
    {Web Counter Page
    You are visitor} count {to this page!

    }
]
```

Le script en exemple ci-dessus charge et sauvegarde le compteur via un fichier texte. Pour rendre accessible ce fichier, il est nécessaire de lui donner les droits appropriés afin de le rendre accessible par tous les utilisateurs.

14.4 Environnement et variables CGI

Lorsqu'un script CGI s'exécute, le serveur fournit des informations à REBOL concernant la requête CGI et ses arguments. Toutes ces informations sont placées sous la forme d'un objet dans l'objet **system/options**. Pour voir les attributs de cet objet, saisissez :

```
probe system/options/cgi
make object! [
    server-software: none
    server-name: none
    gateway-interface: none
    server-protocol: none
    server-port: none
```

```
    request-method: none
    path-info: none
    path-translated: none
    script-name: none
    query-string: none
    remote-host: none
    remote-addr: none
    auth-type: none
    remote-user: none
    remote-ident: none
    Content-Type: none
    content-length: none
    other-headers: []
]
```

Bien sûr, votre script ignorera la plupart de ces informations, mais certaines d'entre elles peuvent être utiles. Par exemple, vous voudrez créer un fichier log qui enregistre les adresses réseau des machines effectuant les requêtes, ou vérifiant le type de navigateur utilisé.

Pour générer une page CGI qui affiche ces informations dans votre navigateur :

```
#!/path/to/rebol -cs

REBOL [Title: "Dump CGI Server Variables"]

print "Content-Type: text/plain^/"

print "Server Variables:"

probe system/options/cgi
```

Si vous voulez utilisez ces informations dans un fichier log, vous devrez les écrire dans un fichier. Par exemple, pour enregistrer les adresses des visiteurs de votre page CGI, vous devrez écrire :

```
write/append/lines %cgi.log
    system/options/cgi/remote-addr
```

Les raffinements **/append** et **/lines** forcent l'écriture à s'effectuer à la suite des enregistrements précédents et ligne par ligne. Voici une autre approche qui inscrit plusieurs items sur la même ligne :

```
write/append %cgi.log reform [
    system/options/cgi/remote-addr
    system/options/cgi/remote-ident
    system/options/cgi/content-type
    newline
]
```

14.5 Requêtes CGI

Il existe deux méthodes pour fournir des données à votre script CGI : GET et POST. La méthode GET encode les données CGI dans l'URL. Elle est utilisée pour fournir des informations au serveur. Vous aurez remarqué sans doute que certaines URLs ressemblent à celle-ci :

```
http://www.example.com/cgi-bin/test.r?&data=test
```

La chaîne de caractères qui suit le point d'interrogation (?) fournit les arguments au script CGI. Parfois, ceux-ci peuvent être assez longs. La chaîne est passée à votre script lorsque celui s'exécute. Elle peut être obtenue à partir de l'attribut **cgi/query-string**.

Par exemple, pour afficher cette chaîne depuis un script :

```
print system/options/cgi/query-string
```

Les données contenues dans la chaîne peuvent inclure n'importe quelle data dont vous avez besoin. Cependant, parce que cette chaîne fait partie de l'URL, les données doivent y être encodées. Il y a des restrictions sur les caractères qui y sont autorisés.

De plus, lorsque ces données sont produites par un formulaire HTML, elles sont encodées de façon standard. Ces données peuvent être décodées et mises dans un objet avec le code :

```
cgi: make object! decode-cgi-query
    system/options/cgi/query-string
```

La fonction **decode-cgi-query** renvoie un bloc qui contient les noms des variables et leurs valeurs. Voir l'exemple avec le formulaire HTML dans la section suivante.

La méthode POST transmet les données CGI sous forme d'une chaîne. Les données n'ont pas besoin d'être encodées. Elles peuvent être dans n'importe quel format que vous le souhaitez et peuvent même être binaires. Les données sont lues à partir de l'entrée standard. Vous devrez les lire depuis l'entrée standard avec un code comme celui-ci :

```
data: make string! 2002
read-io system/ports/input data 2000
```

Ceci devrait sélectionner les 2000 premiers octets de données POST et les mettre dans une chaîne de caractères.

Une bonne pratique pour les données POST est d'utiliser un dialecte REBOL et de créer un petit analyseur (parseur). Les données POST peuvent être chargées et analysées sous forme de bloc. Voir le chapitre sur le Parsing.

Avertissement au sujet des blocs :

Ce n'est pas une bonne idée de passer à REBOL des blocs pour qu'ils soient directement évalués, car cela induit un risque sur la sécurité. Par exemple, quelqu'un pourrait envoyer via POST un bloc qui permettrait de lire ou d'écrire des fichiers sur le serveur. Par ailleurs, passer des blocs qui sont

interprétés par votre script (via un dialecte) est sans danger.

Voici un exemple de script qui affiche les datas POST dans votre navigateur :

```
#!/path/to/rebol -cs

REBOL [Title: "Show POST data"]

print "Content-Type: text/html^/"
data: make string! 10000
foreach line copy system/ports/input [
    repend data [line newline]
]

print [

    {Here is the posted data.}
    ----data

]
```

14.6 Traitement des formulaires HTML

Le protocole CGI est utilisé souvent afin de traiter des formulaires HTML. Les formulaires acceptent des champs variés et les soumettent au serveur Web avec la méthode GET ou la méthode POST.

Voici un exemple qui utilise la méthode GET du CGI pour traiter un formulaire et envoyer un email en guise de résultat. Il y a deux parties : la page HTML et le script CGI.

Voici une page HTML qui comprend un formulaire :

```
CGI Emailer----

Enter your email address:




Enter message here.
```

Lorsque le formulaire HTML est validé, il est traité par la méthode GET et les datas passées au script **send.r**. Voici un exemple de script. Ce script décode les données du formulaire et envoie le message électronique. Il retourne une page de confirmation.

```
 #!/path/to/rebol -cs

 REBOL [Title: "Send CGI Email"]

 print "Content-Type: text/html^/"

 cgi: make object! decode-cgi-query
     system/options/cgi/query-string

 print {Email Status----
}

 failed: error? try [send to-email cgi/email cgi/message]

 print either failed [
     {The email could not be sent.}
 ][
     [{The email to} cgi/email {was sent.}]
 ]

 print {}
```

Ce script doit être nommé send.r et être placé dans le répertoire cgi-bin. Ses droits doivent permettrent qu'il soit lu et exécuté par tous.

Lorsque le formulaire HTML aura été soumis par un navigateur, le script s'exécutera. Il décode la chaîne de caractères de la requête CGI dans l'objet **cgi**. L'objet a comme variables l'email et le message qui sont utilisés par la fonction **send**. Avant l'envoi du courrier électronique, le champ "email", de type *string*, est converti en type de données *email*. La fonction **send** est placée à l'intérieur d'un bloc **try** pour capturer les erreurs pouvant se produire, et le message qui est généré dans ce cas.

D'autres exemples de scripts CGI peuvent être trouvés dans la bibliothèque de scripts REBOL : http://www.rebol.com/library/library.html.

15 TCP - Transmission Control Protocol

En supplément de tous les protocoles décrits précédemment, vous pouvez créer vos propres serveurs et clients avec le protocole TCP (Transmission Control Protocol).

15.1 Créer des clients

Des ports TCP peuvent être ouverts avec REBOL de la même façon qu'avec les autres protocoles, en utilisant une URL TCP. Pour ouvrir une connexion TCP vers un serveur Web (HTTP), sur le port 80 :

```
http-port: open tcp://www.example.com:80
```

Une autre manière d'ouvrir une connexion TCP est de fournir les spécifications de port directement. Cela remplace l'ouverture d'une URL et est souvent plus commode :

```
http-port: open [
    scheme: 'tcp
    host: "www.example.com"
    port-id: 80
]
```

Puisque les ports sont des séries, vous pouvez utilisez les fonctions relatives aux séries pour émettre et recevoir des données. L'exemple ci-dessous effectue une requête sur le serveur HTTP ouvert dans l'exemple précédent. Il utilise la fonction **insert** pour placer des datas dans la série, le port http-port qui les envoie au serveur :

```
insert http-port join "GET / HTTP/1.0^/^/"
```

Les deux caractères de nouvelle ligne (^/) sont utilisés pour signifier au serveur que l'en-tête HTTP a été émis.

Les caractères (^/ ou newline) sont automatiquement convertis en séquences CR LF car le port a été ouvert en mode texte. Le serveur traite la requête HTTP et retourne un résultat au port. Pour lire le résultat, utilisez la fonction **copy** :

```
while [data: copy http-port] [prin data]
```

Cette boucle continuera de récupérer les datas jusqu'à ce que **none** soit retourné par la fonction **copy**. Ce comportement diffère selon les protocoles. Un "**none**" est retourné car le serveur ferme la connexion. D'autres protocoles peuvent utiliser un caractère spécial pour marquer la fin du transfert.

A présent que toutes les datas ont été reçues, le port HTTP doit être fermé :

```
close http-port
```

Voici un autre exemple qui ouvre un port POP en TCP sur un serveur :

```
pop: open/lines tcp://fred.example.com:110
```

Cet exemple utilise le raffinement **/lines**. La connexion sera à présent orientée lignes. Les données seront lues et écrites sous forme de lignes.

Pour lire la première ligne depuis le serveur :

```
print first pop
+OK QPOP (version 2.53) at fred.example.com starting.
```

Pour émettre vers le serveur un nom d'utilisateur pour le login POP :

```
insert pop "user carl"
```

Puisque le port est ouvert en mode ligne, un caractère de fin de ligne est émis aprés la chaîne "user carl" insérée.

La réponse du serveur POP peut être lue avec :

```
print first pop
+OK Password required for carl.
```

Et le reste de la communication devrait s'effectuer :

```
insert pop "pass secret"

print first pop
+OK carl has 0 messages (0 octets).
insert pop "quit"

first pop
+OK Pop server at fred.example.com signing off.
```

La connexion doit enfin être fermée :

```
close pop
```

15.2 Création de serveurs

Pour créer un serveur, vous devrez attendre des demandes de connexions et y répondre lorsqu'elles se produisent. Pour définir un port sur votre machine qui peut utilisé pour attendre les connexions entrantes :

```
listen: open tcp://:8001
```

Remarquez que vous n'indiquez pas de nom de machine, seulement un numéro de port. Ce type de port est appelé un port d'écoute (listen port). Votre système accepte maintenant les connexions sur le port numéro 8001.

Pour attendre une connexion d'une autre machine, vous attendez sur le port d'écoute.

```
wait listen
```

Cette fonction ne se terminera pas tant qu'une connexion n'aura pas été réalisée.

NOTE : Il existe diverses options possibles pour wait. Par exemple, vous pouvez attendre sur plusieurs ports ou aussi avec un time-out.

Vous pouvez maintenant ouvrir le port pour la connexion depuis la machine qui a contacté votre système.

```
connexion: first listen
```

Ceci renvoie la connexion qui a été faite sur le port d'écoute. C'est un port comme tous les autres, et il peut être utilisé pour émettre et recevoir des données, au moyen des fonctions **insert**, **copy**, **first**, et des autres fonctions relatives aux séries.

```
insert connexion "you are connected^/"

while [newline  char: first connexion] [
    print char
]
```

Lorsque la communication est complète, la connexion doit être fermée.

```
close connexion
```

Vous êtes à présent prêt pour la connexion suivante sur le port d'écoute. Vous pouvez attendre encore et utiliser **first** encore pour la nouvelle connexion.

Lorsque votre serveur en a fini, vous devez fermer le port d'écoute avec :

```
close listen
```

15.3 Un tout petit serveur

Voici un serveur REBOL assez commode qui nécessite seulement que quelques lignes de code. Ce serveur évalue le code REBOL qui lui est envoyé. Les lignes de REBOL en provenance d'un client sont lues jusqu'à ce qu'une erreur se produise. Chaque ligne doit être une expression REBOL compléte. Elle peut être de n'importe quelle longueur mais doit faire une seule ligne.

```
server-port: open/lines tcp://:4321

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

```
    ]
    close connexion-port
]
close server-port
```

Si une erreur se produit, la connexion est fermée et le serveur se met en attente de la connexion suivante.

Voici un exemple de script pour un client qui vous permet de rentrer à distance des commandes REBOL :

```
server: open/lines tcp://localhost:4321
until [error? try [insert server ask "R> "]]
close server
```

Ici la requête est faite pour déterminer si la connexion a été interrompue du fait d'une erreur.

15.4 Test du code TCP

Pour tester le code de votre serveur, connectez-vous depuis votre propre machine, plutôt que d'avoir un serveur et un client. Ceci peut être fait avec deux processus REBOL distincts ou même un seul processus.

Pour vous connecter en local sur votre machine, vous pouvez utilisez une ligne comme celle-ci :

```
port: open tcp://localhost:8001
```

Voici un exemple qui crée deux ports connectés entre eux en mode ligne. Il s'agit d'une sorte de port "echo" puisque vous émettez des datas vers vous-mêmes. C'est un bon test pour votre code et l'usage du réseau :

```
listen: open/lines tcp://:8001
remote: open/lines tcp://localhost:8001
local: first listen
insert local "How are you?"
print first remote  ; response
close local
close remote
close listen
```

16 UDP - User Datagram Protocol

Le protocole UDP (User Datagram Protocol) est un autre protocole de transport qui fournit une méthode pour communiquer entre machines, mais qui n'est pas orienté connexion, à la différence de TCP.

Il permet d'envoyer des datagrammes, des paquets de données entre des machines. L'utilisation d'UDP est assez différente de celle de TCP. Le protocole UDP est plus simple, mais il est aussi moins sûr. Il n'y a pas de garantie qu'un paquet atteigne toujours sa destination. De plus, UDP ne dispose pas de mécanisme de contrôle de flux. Si vous envoyez des messages trop rapidement, des paquets peuvent être perdus.

Comme pour TCP, la fonction **wait** peut être utilisée pour attendre qu'un nouveau paquet arrive et la fonction **copy** peut être utilisée pour renvoyer les données. S'il n'y a pas de données, **copy** attend jusqu'à ce qu'il y en ait. Notez cependant que la fonction **insert** n'attend jamais.

Voici un exemple de script pour un petit serveur UDP :

```
udp: open udp://:9999
wait udp
print copy udp
insert udp "response"
close udp
```

Les messages insérés dans le port *udp* ici par le serveur sont expédiés vers le client de qui a été reçu le dernier message. Ceci permet aux réponses d'être émises pour les messages entrants. Cependant, contrairement à TCP, vous n'avez pas de connexion continue entre les machines. Chaque transfert de paquet donne lieu à un échange spécifique.

Le script client pour communiquer avec le serveur ci-dessus pourrait être :

```
udp: open udp://localhost:9999
insert udp "Test"
wait udp
print copy udp
close udp
```

Vous devez aussi savoir que la taille maximale d'un paquet UDP dépend du système d'exploitation. 32 Ko et 64 Ko représentent des valeurs courantes. Afin d'envoyer de grandes quantités de données, vous devrez mettre en buffer les datas, et les diviser en paquets plus petits. Par ailleurs, votre programmation doit être particulièrement soignée pour être certain que chaque partie des données est bien reçue. Rappelez-vous qu'avec UDP, il n'y a pas de garantie.