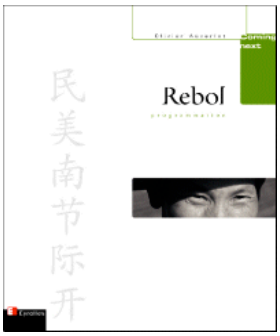
Chapitre 18

Développer des applications pour Rebol/IOS



Ce chapitre inédit aurait du figurer dans la réédition du livre "Programmation Rebol" (ISBN: 2-212-11017-0) publié par les éditions Eyrolles. Ce projet ayant été abandonné, le texte est versé au RDP (Rebol Documentation Project).

Auteur: Olivier Auverlot
Dernière modification: 14 mars 2005
Nombre de pages: 43

Rebol/IOS est à la fois un outil de travail et une plateforme de développement, qui fournit aux programmeurs l'ensemble des utilitaires et des fonctions nécessaires au développement de logiciels.

La combinaison de ces deux aspects en fait un produit adaptable à toutes les situations et extensible à l'infini.

Les bases de la programmation distribuée avec IOS

Vous entrez ici de plain-pied dans le domaine passionnant de la programmation distribuée.

Contrairement aux technologies Web classiques, dans lesquelles le client n'est qu'un périphérique destiné à l'affichage et à la saisie, le serveur ayant la pleine charge du traitement et de la fourniture des données, Rebol/IOS s'appuie sur une répartition intelligente du travail entre le serveur Express et le client Link. Tous deux travaillent de concert, et c'est généralement à Link que revient la tâche de traiter les données que lui a fournies Express.

Cette organisation permet de concevoir des applications ayant un degré de réactivité ainsi qu'une ergonomie proches des applications bureautiques et, surtout, très économes en ressources réseau.

Les applications Web que l'on peut considérer comme appartenant à l'ancienne génération (CGI, PHP, JSP, servlet, ASP, ASP.Net, etc.) rendent obligatoire la communication avec le serveur HTTP chaque fois que l'utilisateur change de page HTML en validant, par exemple, une opération. De plus, l'ergonomie de ces applications est souvent limitée par l'usage du HTML et des contraintes qu'impose la recherche de la compatibilité avec un maximum de navigateurs.

Avec Rebol/IOS, tout est différent. Les applications sont des reblets téléchargées sur le poste client et sauvegardées dans son cache. Lorsque l'utilisateur travaille avec l'une d'entre elles, celle-ci est disponible instantanément, qu'il soit ou non connecté au serveur Express. Il utilise alors une véritable application locale, réagissant instantanément à ses sollicitations et disposant d'une interface fondée sur VID, dont le comportement et les fonctionnalités sont proches de l'environnement graphique qu'il utilise quotidiennement avec des produits bureautiques.

Avec une reblet, les communications vers le serveur Express sont réduites au stricte minimum et sont destinées uniquement à l'envoi et à la réception de données stockées sur le serveur ou obtenues par son intermédiaire. Tout cela est réalisé en toute sécurité puisque les échanges en Link et Express sont toujours cryptés.

En permettant la conception d'applications dont l'administration et la diffusion sont centralisées mais dont l'exécution est laissée en grande partie au poste client, Rebol/IOS conjugue les avantages de deux univers jusqu'ici antagonistes.

Les différents types d'applications

Lorsque vous développez pour Rebol/IOS, vous disposez de trois types d'architectures pour vos applications. Cette flexibilité est surtout dictée par un souci de sécurité. Elle respecte en fait la gestion des droits de chaque utilisateur. Vous pouvez publier un script Rebol dont l'exécution sera limitée au poste client ou développer une reblet utilisant les spécificités de Rebol/Link ou encore concevoir une application distribuée dans laquelle les traitements sont répartis entre le client et le serveur Express.

Publier un script Rebol

La méthode la plus simple pour ajouter de nouvelles applications à Rebol/IOS consiste à publier un script Rebol. Cela revient à publier un fichier et implique donc que vous ayez le droit New-App activé dans votre profil utilisateur. Ce droit est normalement actif pour l'ensemble des utilisateurs, puisqu'il permet la publication de documents et autorise la modification du contenu d'un fileset existant.

Il n'est guère utile de revenir sur la méthodologie de publication d'un document, puisqu'elle a été présentée au chapitre précédent. Comme vous pouvez le constater, ce type de déploiement d'une application est extrêmement simple mais présente de nombreuses limitations :

- Le script est obligatoirement ajouté à un fileset existant.

- Il n'est pas possible de spécifier des paramètres particuliers pour le script puisqu'il hérite des attributs du fileset auquel il a été ajouté. L'icône apparaissant sur le bureau Link est l'icône standard représentant un script Rebol ou, au mieux, une des icônes déjà présentes sur le serveur. Les règles de synchronisation ne sont pas modifiables, et le script doit suivre le comportement du fileset auquel il appartient.
- Si l'accès au script doit être limité à certains utilisateurs, vous devez le placer dans un dossier privé.
- Le script ne peut pas utiliser les fonctionnalités spécifiques du client Link concernant la communication avec l'environnement d'exécution local et l'échange de données avec le serveur Express.

La figure 18-1 illustre la publication de divers scripts Rebol dans le répertoire de l'utilisateur Olivier.

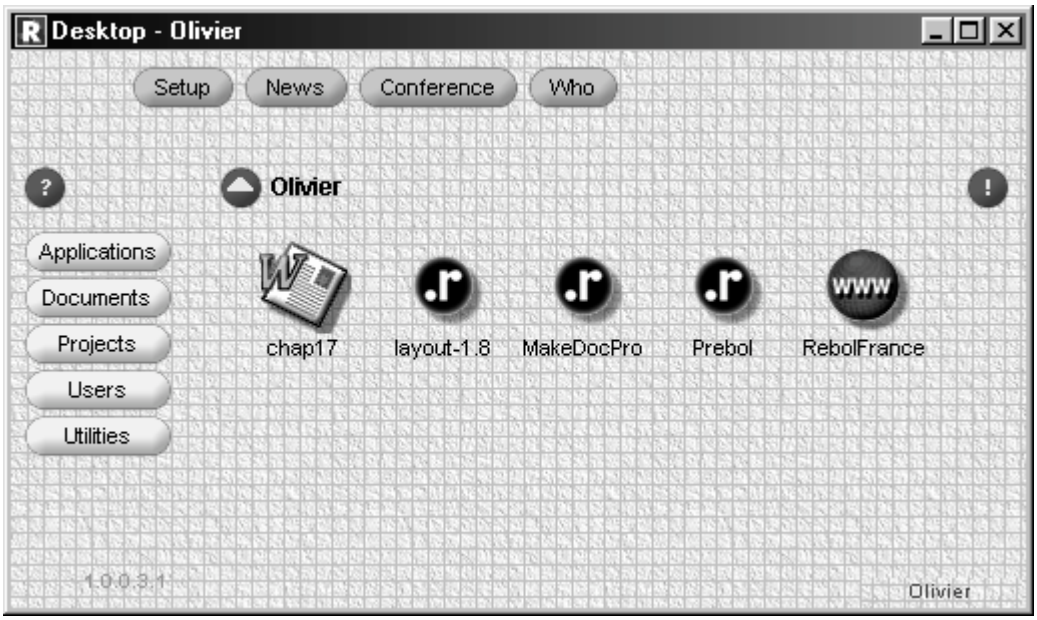


Figure 18-1 Différents scripts publiés par l'utilisateur Olivier

Cette méthode permet d'intégrer à Rebol/IOS des scripts conçus pour Rebol/Core ou Rebol/View sans que cette opération nécessite la moindre modification des applications. Pour créer des véritables reblets pour Rebol/IOS, vous devez utiliser les fonctionnalités particulières de ce serveur d'applications, réparties entre le client Link et le serveur Express.

Développer une reblet pour Link

Pour exploiter certaines particularités du client Rebol/Link, vous devez spécifier dans son en-tête que la reblet exécutée est un script destiné à être exécuté dans l'environnement Rebol/IOS. Il vous suffit d'indiquer que le type du script est 'app-link pour avoir accès à de nombreuses extensions du dictionnaire de

l'évaluateur. Votre script devient alors spécifique à Rebol/IOS et ne peut plus être utilisé avec Rebol/Core, Rebol/View ou même Rebol/Command.

Ces nouvelles fonctions concernent principalement :

- L'obtention d'informations sur le compte utilisateur courant.
- L'intégration de Rebol/Link sur la machine hôte.
- L'exploitation des données présentes dans la copie locale de la base de registre de Rebol/IOS. Le client Link peut extraire des informations sur les différents filesets et fichiers utilisables.

La différence fondamentale entre un script publié et une véritable reblet réside dans le fait que cette dernière peut être véritablement installée sur le serveur Express à l'aide d'un fileset spécifique. Vous pouvez de la sorte contrôler avec précision les paramètres de synchronisation et de partage, ainsi que les attributs de présentation (titre, icône, position dans l'arborescence du serveur, etc.).

Pour développer une telle reblet, l'utilisateur a besoin de l'activation du droit New-App dans son profil. Il doit en effet pouvoir créer ou modifier un fileset. Tout comme un script publié, une reblet pour Link est copiée sur le serveur pour être diffusée aux clients, mais son exécution est intégralement confiée à ces derniers. Pour réaliser de véritables programmes distribués, vous devez développer des applications IOS.

Concevoir une application IOS

Une application IOS est constituée des trois composants suivants :

- Un fileset contenant les différents éléments applicatifs et éventuellement les données manipulées par l'application. Un ou plusieurs filese*ts* supplémentaires peuvent être utilisés pour stocker les informations.
- Une ou plusieurs reblets exécutées par le poste client et dont le type est 'Link-app. Ces programmes ont la responsabilité de l'aspect présentation, ainsi que de la communication avec l'utilisateur et du traitement des données.
- Une portion de code située sur le serveur dans le fileset principal de l'application. Ses fonctions consistent à réaliser des traitements spécifiques sur le serveur à la demande de la reblet et à fournir des données.

Pour d'évidentes raisons de sécurité, l'autorisation d'écrire du code serveur est soumise à l'activation du droit Serve-Code dans le profil de l'utilisateur. Une application IOS permet de la sorte la répartition du travail entre le poste client et le serveur ainsi que le stockage de données à l'aide des filesets. Cette architecture particulière impose l'usage d'une méthodologie de développement spécifique.

Méthodologie de développement

Pour développer une reblet Link ou une application IOS, il vous faut suivre une méthodologie particulière, qui bouscule nombre d'habitudes.

La grande idée est que vous ne travaillez jamais directement sur le serveur Express. Avec les technologies Web classiques, vous avez probablement pris l'habitude de transférer vos pages HTML et vos applications sur un serveur

HTTP à l'aide d'un client FTP ou d'une simple disquette. Avec Rebol/IOS, chaque client Rebol/Link est un bureau virtuel. C'est par son intermédiaire que vous installez et mettez à jour vos reblet*s* sur le serveur Express.

Link est sécurisé et vous autorise à travailler partout. À l'aide d'une connexion Internet quelconque, vous avez la possibilité de transférer vos applications sur un serveur Express.

Pour envoyer la définition du fileset de l'application ainsi que la reblet pour le client et le code optionnel pour le serveur, le principe retenu est celui d'un script d'installation.

Un projet se compose au minimum de deux fichiers :

- Le code de la reblet destinée au client, ayant le type 'Link-app.
- Un script d'installation contenant les attributs du fileset, la liste des fichiers contenus par le fileset, parmi lesquels la reblet cliente qui doit être synchronisée et diffusée aux clients, et le code éventuellement exécuté par le serveur.

L'idéal est de créer un répertoire par projet. Celui-ci contient la reblet et le script d'installation. Pour ce dernier, vous pouvez prendre la bonne habitude de lui attribuer un nom commençant par "install-" et se terminant par le nom de la reblet. Cela vous permet de le repérer aisément parmi les différents fichiers de votre projet. Vous pouvez également placer dans ce répertoire des fichiers additionnels, tels que l'icône qui sera affichée sur le bureau Link.

Pour implanter ou mettre à jour votre programme sur le serveur, il vous suffit d'utiliser la séquence de touche **CTRL + L** et de sélectionner le script d'installation. Le client Link exécute les différentes opérations. S'ils sont connectés, les utilisateurs ou les membres des groupes avec qui vous partagez votre application reçoivent immédiatement la dernière version de votre produit.

C'est aussi simple que cela. Vous allez maintenant découvrir par la pratique l'aisance que procure le serveur d'applications Rebol/IOS.

User-infos, votre première reblet

Pour ce premier projet destiné à Rebol/IOS, vous allez réaliser une reblet Link nommée User-Infos, dont la fonction consiste à afficher quelques informations sur le compte de l'utilisateur courant ainsi que sur l'installation de Link.

Vous écrirez votre première reblet pour IOS, définirez un fileset et concevrez le script d'installation.

Présentation et écriture du script

La reblet User-Infos doit afficher dans une fenêtre le nom de l'utilisateur et son adresse e-mail. Pour obtenir ces données, vous disposez de l'objet user-prefs, dont les propriétés rassemblent toutes les informations utiles sur l'utilisateur. Vous utiliserez les propriétés user-prefs/name et user-prefs/email.

L'écriture du code de la reblet ne pose pas de grandes difficultés. L'en-tête doit simplement préciser à l'aide de l'attribut Type qu'il s'agit d'une application pour Rebol/IOS :

```
REBOL [
    Title: "user-infos"
    Type: 'link-app
]
```

Vous devez ensuite définir un layout pour l'affichage. Celui-ci contient la définition des styles label et info, destinés respectivement à afficher l'intitulé de l'information et la valeur obtenue.

Un bouton est défini afin de permettre à l'utilisateur de fermer proprement l'application. Le style utilisé pour cela est btn. Il s'agit d'un nouveau type de bouton introduit par Link et dont les coins son arrondis :

```
main: layout [
    style label text 150 'right
    style info text 200
    across
    label "Utilisateur:" nom: info return
    label "email:" email: info return
    btn "Quitter" [ quit ]
]
```

Il ne vous reste plus qu'à initialiser les champs nom et info à l'aide des données présentes dans l'objet user-prefs pour que la fenêtre s'affiche au centre de l'écran :

```
nom/text: user-prefs/name
email/text: user-prefs/email
view/title center-face main "Informations sur le compte actif"
```

Pour sauvegarder les différents fichiers composant votre projet, créez un répertoire user-infos. Vous devez enregistrer le script que vous venez de rédiger dans ce répertoire en le nommant user-infos.r.

La première étape est terminée. Il vous faut maintenant définir le fileset auquel appartiendra cette reblet.

Créer le fileset

Pour créer un fileset et copier les fichiers constituant l'application, vous devez rédiger un script de type link-app.

Il est recommandé de désactiver le gestionnaire de sécurité afin d'éviter que la procédure d'installation ne vous demande de confirmer l'envoi de chacun des fichiers présents dans le fileset :

```
REBOL [
    title: "install-user-infos"
    Type: 'link-app
]
secure none
```

Rebol/Link dispose du mot install-fileset, dont le seul paramètre est un bloc contenant les différents attributs du nouveau fileset. Ce dernier, qui a pour nom user-infos, est composé des deux fichiers suivants :

- Le script de l'application user-infos.r.
- L'icône user-infos.gif devant être affichée sur le bureau Link. Une icône peut être aux formats GIF, JPEG, PNG ou BMP. Par défaut, la taille standard est de 48 × 48 pixels.

Ces deux fichiers sont référencés dans l'attribut files, qui est une liste de blocs. Le premier élément de chaque bloc correspond à l'endroit où sera recopié le fichier indiqué par le second élément. Le répertoire apps utilisé est en fait le classeur Applications du bureau Link.

L'attribut tags contient différents paramètres concernant surtout les droits d'accès et d'usage du fileset :

- Users contient la liste des utilisateurs ayant accès au fileset.
- Groups contient la liste des groupes d'utilisateurs ayant accès au fileset.
- Access spécifie les droits des utilisateurs autorisés par les paramètres Users et Groups. Pour cette application, admin et olivier peuvent modifier les droits d'accès au fileset (rights), changer la valeur des tags version, install-date et priority (properties), modifier la liste des fichiers appartenant au fileset (change) et détruire le fileset (delete).
- Avec l'attribut icons, vous spécifiez le nom de l'application (name), le chemin d'accès à l'image (image), le répertoire où apparaît l'icône (folder), le texte affiché lorsque le pointeur de la souris passe sur l'icône (fileset:info) et le fichier appelé lorsque l'utilisateur sélectionne l'icône (item).

Lorsque votre fileset est entièrement défini, le script d'installation doit avoir l'aspect suivant :

```
REBOL [
title: "install-user-infos"
Type: 'link-app
]

secure none

install-fileset [
    fileset: 'user-infos

    tags: [
        users: [ "admin" "olivier" ]
        groups: [ ]
        access: [
            properties: rights: delete: change: [ "admin" "olivier"
]
        ]
        icons: [
            [
                name: "User-Infos"
                item: %apps/user-infos/user-infos.r
                folder: %apps/
                image: %desktop/icons/user-infos.gif
                info: "Informations sur le compte actif"
            ]
        ]
    ]

    files: [
        [%apps/user-infos/user-infos.r %user-infos.r]
        [ %desktop/icons/user-infos.gif %user-infos.gif ]

    ]
]
```

Il ne vous reste qu'à enregistrer ce fichier dans le répertoire user-infos en lui donnant pour nom install-user-infos.r.

Copier les fichiers sur le serveur

La dernière étape consiste à démarrer la procédure d'installation.

Avant cela, assurez-vous que les deux scripts et le fichier contenant l'icône sont dans le même répertoire. À l'aide de votre client Link, pressez les touches **CTRL + L,** et sélectionnez le fichier install-user-infos.r. Le gestionnaire de sécurité vous demande l'autorisation de se désactiver. Suite à votre confirmation, les fichiers user-infos.r et user-infos.gif sont copiés dans le nouveau fileset.

La figure 18-2 illustre l'installation du nouveau fileset sur le serveur Express.

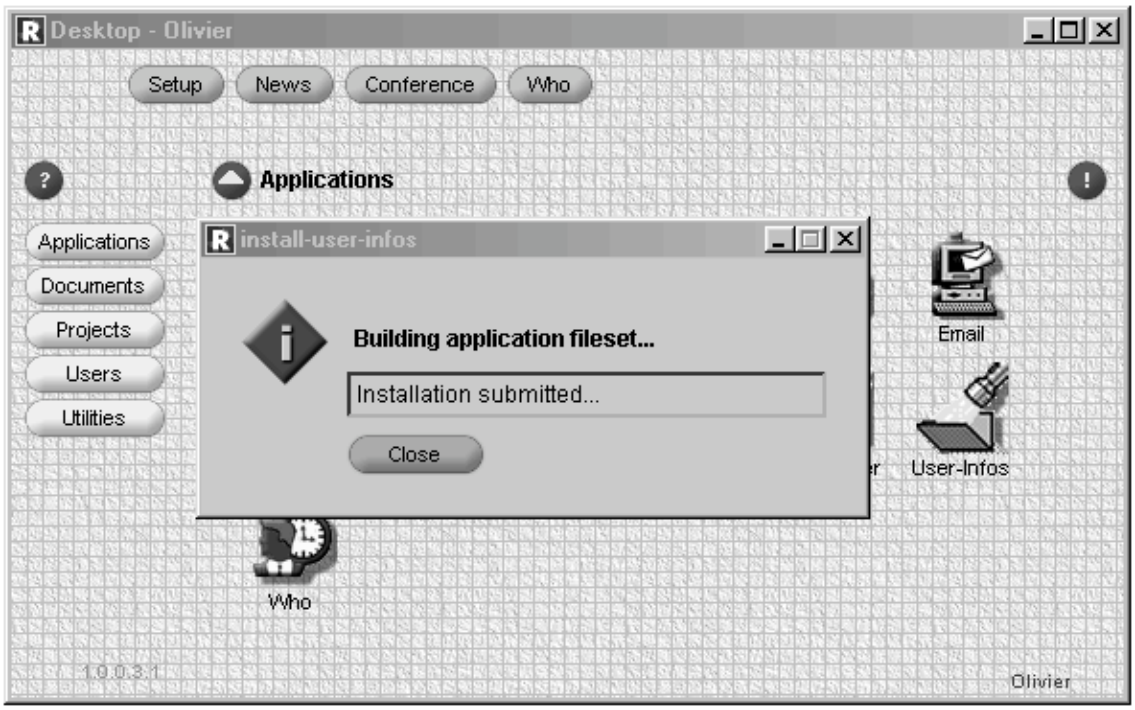


Figure 18-2 L'installation de l'application terminée

Pour les utilisateurs admin et olivier, une nouvelle icône apparaît parmi les applications de Rebol/Link. Un simple clic gauche sur l'image suffit pour lancer l'application.

La figure 18-3 illustre l'exécution de votre nouvelle reblet.

Figure 18-3 Exécution de la reblet user-infos

Vous venez de définir votre premier fileset et de créer votre première véritable reblet pour Rebol/IOS. Comme vous avez pu le constater, ces deux éléments sont étroitement liés, et le développement pour IOS implique une parfaite connaissance du fonctionnement des filesets.

Vous allez maintenant approfondir vos connaissances sur ces sujets en étudiant les fonctionnalités spécifiques de Link, telles que l'exploration de la base de registre locale de Rebol/IOS et la gestion des événements.

Les fonctionnalités de Rebol/Link

Afin de faciliter le développement de reblets, Rebol/Link met à la disposition du programmeur un nombre important de nouveaux mots. Ces mots permettent d'obtenir des informations sur l'état du poste client ainsi que de manipuler les filesets locaux et de gérer les événements.

Obtenir des informations sur le client Link

Le dictionnaire de Link contient plusieurs mots permettant d'obtenir des informations sur le statut du client. Vous pouvez, par exemple, savoir si Rebol/Link est connecté à un serveur Express, connaître le répertoire de travail de Rebol/Link ou encore obtenir la liste des reblets lancées sur le poste client.

Ces fonctionnalités sont précieuses puisqu'elles permettent à une reblet de s'adapter à la configuration de la machine hôte lors de son exécution.

Link est-il connecté à un serveur Express ?

À l'aide du mot connected?, une reblet peut savoir si elle est connectée avec un serveur Express. Ce mot renvoie une valeur booléenne selon que le client est on-line ou off-line :

```
REBOL [
    subject: "test de la connexion"
    Type: 'link-app
]

either connected? [
    alert "Vous êtes connecté au serveur Express"
] [ alert "Vous n'êtes pas connecté" ]
```

Le mot connected? est très important car il vous permet d'adapter le comportement de votre reblet selon que Link est connecté ou qu'il travaille en mode local.

Dans certains cas, vous pouvez de la sorte désactiver des traitements exigeant l'échange d'informations avec le serveur ou encore décider de stocker provisoirement les données sur la machine hôte afin de reporter à plus tard la synchronisation avec le serveur Express.

Le répertoire de travail de Link

Pour savoir où se trouve le répertoire de travail de Rebol/Link sur le poste client, vous disposez du mot link-root. Le chemin d'accès obtenu est composé du répertoire d'installation de Rebol/Link et du répertoire contenant les données du serveur Express utilisé, comme illustré à la figure 18-4.

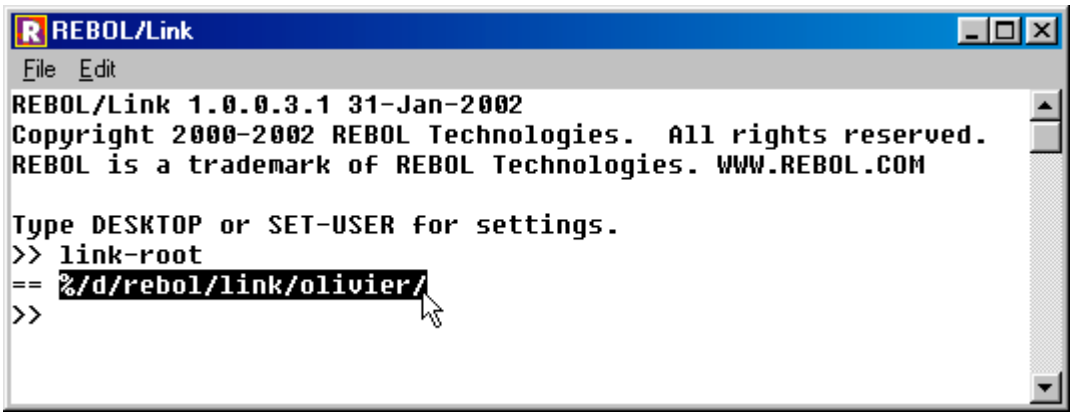


Figure 18-4 Le mot link-root retournant le répertoire de travail de Link

Lorsque Link lit ou écrit des fichiers sur le disque dur de la machine hôte, il utilise de préférence l'arborescence présente dans son répertoire d'installation. Cela permet, par exemple, de lire le contenu des fichiers synchronisés et stockés dans le répertoire portant le même nom que le serveur utilisé.

Obtenir des informations sur l'installation de Link

Avec l'objet user-prefs présenté précédemment, vous pouvez obtenir quantité d'informations sur les paramètres d'installation de Rebol/Link.

Le tableau 18-1 récapitule les principales propriétés de user-prefs.

Tableau 18-1 Principales propriétés de l'objet user-prefs

PROPRIETE	DESCRIPTION
Name	Compte de l'utilisateur ayant installé link
Pass	Mot de passe crypté de l'utilisateur
Express	Nom du serveur express utilisé
Server	Nom de la machine hébergeant le serveur express
Script	Chemin d'accès au proxy IOS
Proxy-user	Compte de l'utilisateur sur le proxy HTTP
proxy-pass	Mot de passe de l'utilisateur sur le proxy HTTP
Root	Chemin d'accès au cache de Link
Log	Activation du journal de Link
Debug	Suivi des erreurs d'exécution de Link
auto-connect	Connexion automatique de Link au démarrage
Desktop	Affichage du bureau au lancement de Link
Skin	Modification de l'apparence de Link
Location	Localisation de l'utilisateur
Status	Activité de l'utilisateur
Passive-ftp	Utilisation du mode FTP passif
Email	Adresse e-mail de l'utilisateur
smtp-server	Serveur SMTP de l'utilisateur
pop-server	Serveur POP de l'utilisateur
proxy-host	Nom du serveur proxy HTTP
proxy-port	Port TCP du serveur proxy HTTP
proxy-type	Type du serveur proxy HTTP

Soyez prudent en utilisant cet objet, car il s'agit des préférences de l'utilisateur ayant installé Link sur la machine hôte. Certaines informations ne sont donc pas forcément exploitables si plusieurs utilisateurs utilisent la même machine ou que l'utilisateur dispose de plusieurs identités.

Gestion des tâches

Lorsque vous travaillez avec Rebol/Link, vous pouvez exécuter autant de reblets que votre machine peut en supporter. Pour gérer ces différentes tâches, Link intègre un gestionnaire, qui vous permet d'obtenir la liste des applications actives mais également de les lancer ou de les stopper à l'aide des mots start-app et stop-app.

Le mot start-app reçoit en argument le chemin d'accès à la reblet devant être exécutée. Pour le script who, contenu dans le répertoire /apps présent dans l'arborescence de Link et stocké dans le fichier who.r, il vous suffit de saisir la commande suivante :

```
start-app join link-root "/apps/who.r"
```

L'arrêt est un peu plus complexe. En effet, la commande stop-app doit recevoir en paramètre une valeur numérique, qui correspond au numéro de la tâche attribué à l'application par Link. Pour obtenir cet app-id ou encore connaître la liste des tâches, vous devez demander ces informations au client Link au moyen d'un ensemble de commandes composant son dialecte d'administration.

Utiliser les commandes du client

Rebol/Link dispose d'un dialecte dédié à son administration et à son exploitation. À une exception près, les commandes disponibles permettent principalement d'obtenir des informations sur le fonctionnement du client.

Pour utiliser ce dialecte, le principe retenu est celui de l'envoi de message. À la fois expéditeur et destinataire, Rebol/Link se transmet la commande à l'aide des mots get-link et send-link. Ces deux mots ne permettent pas l'utilisation des mêmes commandes et surtout fonctionnent sur un modèle événementiel totalement différent.

Questionner et attendre une réponse avec get-link

Le mot get-link transmet une commande et attend son résultat. Il s'agit d'un mode bloquant. Tant que les informations ne sont pas obtenues, l'exécution du script est suspendue.

Ce mot comporte deux paramètres, la commande transmise au client et la fonction appelée lorsque l'opération est achevée. Le résultat est transmis à l'aide du paramètre de cette fonction. Si vous voulez lire la copie locale de la base de registre IOS à l'aide de la commande app-reg, il vous suffit d'écrire le code suivant :

```
get-link 'app-reg func [ data ] [ print mold data ]
```

Ce code peut être rédigé différemment en utilisant une référence à la fonction appelée :

```
mafonction: func [ data ] [
    print mold data
]
get-link 'app-reg :mafonction
```

Si un argument supplémentaire doit être passé à la commande, le mot get-link dispose du raffinement /sub-type. L'exemple suivant récupère les tags décrivant le fileset task-data, dont la fonction est de contenir les données de l'application Taskmaster :

```
mafonction: func [ data ] [
    print mold data
]
get-link/sub-type 'app-tags :mafonction 'task-data
```

Les commandes disponibles sont destinées à obtenir des informations sur l'environnement d'exécution installé sur le poste client.

Les commandes utilisables avec get-link

Les commandes du dialecte d'administration de Link peuvent être réparties en quatre grandes catégories, fournissant des informations sur les filesets, la configuration du client, les tâches actives et les sessions.

Pour les filesets, vous avez déjà découvert app-tags et app-reg, qui retournent respectivement la description d'un fileset et le contenu de la base de registre locale d'IOS. Vous disposez également de la commande file-reg, qui permet d'obtenir la liste des fichiers référencés dans la base de registre locale d'IOS :

```
get-link 'file-reg func [ data ] [ print mold data ]
```

De nombreuses informations concernant la configuration du client peuvent également être obtenues à l'aide de get-link. La commande user-prefs fournit un objet semblable à celui retourné par le mot user-prefs. Les commandes servers et serve-keys vous permettent de connaître la liste des serveurs utilisables avec le client Link et les différentes clés publiques qu'il utilise pour le protocole RMP.

Avec la commande user-accounts, vous obtenez la liste de vos comptes utilisateur pour les différents serveurs Express accessibles :

```
reception: func [ data ] [
    print mold data
```

```
]

get-link 'user-prefs :reception
get-link 'servers :reception
get-link 'server-keys :reception
get-link 'user-accounts :reception
```

Lorsqu'une application est exécutée par Link, celle-ci reçoit un identificateur unique, nommé app-id. Il s'agit d'une valeur entière permettant de distinguer les différentes instances d'une même tâche. Vous pouvez utiliser la commande app-id pour récupérer cet identifiant :

```
get-link 'app-id func [ data ] [ print mold data ]
```

L'historique des applications exécutées est sauvegardé par le client et peut être consulté à l'aide de la commande apps :

```
get-link 'apps func [ data ] [ print mold data ]
```

La commande app-instances donne la liste des instances d'applications en cours d'exécution. Chaque app-id est suivi d'un objet décrivant la tâche active à l'aide de trois propriétés :

- type indique le type de l'application. Si cette propriété contient la valeur 'caller, il s'agit du bureau de Link. Si c'est la valeur 'file, il s'agit de la console de Link. Une reblet est identifiée par la valeur 'doer.
- file-type contient 'link-app si l'application est destinée à Rebol/IOS.
- windows fournit le numéro de la fenêtre utilisée par l'application.

À l'aide de ces commandes, vous pouvez développer rapidement un gestionnaire de tâches pour Rebol/Link. Celui-ci doit afficher la liste des applications actives et vous permettre de stopper l'exécution de chacune d'elles. L'en-tête du script doit indiquer qu'il s'agit d'une application utilisant les spécificités de Rebol/IOS :

```
REBOL [
    subject: "gestionnaire de taches"
    Type: 'link-app
]
```

Pour obtenir la liste des reblets en cours d'exécution, vous devez écrire une fonction lecture-applications utilisant les commandes apps et app-instances. Vous obtenez ainsi la liste des instances et l'historique des tâches.

En utilisant les app-id des applications actives, vous pouvez rechercher les noms des scripts afin de les ajouter au contenu d'un composant text-list de VID. Les différents app-id sont sauvegardés dans une liste nommée lst-app-id :

```
lecture-applications: does [
    get-link 'apps func [ data ] [ historique: copy data ]
    get-link 'app-instances func [ data ] [
        taches: reduce copy data
    ]

    num: none
    lst-app-id: copy []
    lst/lines: copy []
    forskip taches 2 [
        append lst-app-id taches/1
        historique: head historique
        forskip historique 5 [
            if found? find historique/5 taches/1 [
                either historique/2 = 'none [
```

```
                        append lst/lines join to-string second split-
path historique/3 [ " (" taches/1 ")" ]
                ] [ append lst/lines historique/2]
            ]
        ]
    ]
    lst/sld/redrag lst/lc / length? lst/lines
    show lst
]
```

Il ne vous reste qu'à définir et afficher le layout de votre script. Celui-ci contient une liste et deux boutons. Le bouton Quitter a pour unique fonction d'arrêter l'exécution de l'application. Le bouton Stop permet de mettre fin à l'exécution d'une reblet sélectionnée dans la liste. Pour cela, il vous faut récupérer le app-id du script et utiliser la commande stop-app de Link :

```
main: layout [
    lst: text-list 300x100 [ num: face/cnt ]
    across
    btn "Stop" [
        if not none? num [
            stop-app pick lst-app-id num
            lecture-applications
        ]
    ]
    btn "Quitter" [ quit ]
]

lecture-applications
view/title center-face main "Applications actives"
```

La figure 18-5 illustre l'utilisation de votre gestionnaire de tâches.

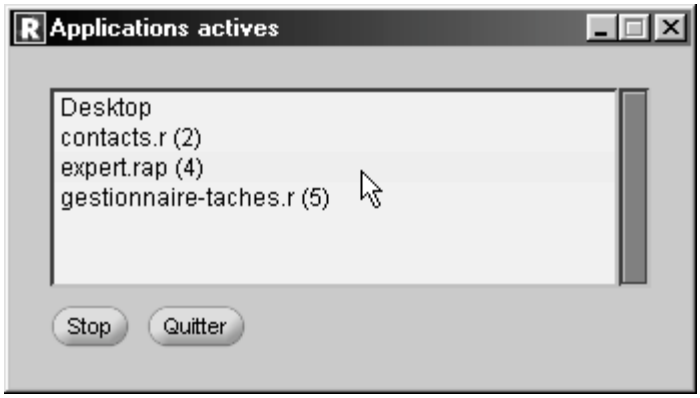


Figure 18-5 Affichage des reblets en cours d'exécution

Rebol/IOS utilise un mécanisme de gestion des sessions extrêmement précis, qui distingue chaque utilisateur et chaque application. Lorsqu'un client Link se connecte à un serveur Express, ce dernier lui attribue un numéro de session unique, qui sera utilisé par la suite pour l'identification du client. De plus, chaque application exécutée par le client est également identifiée par la combinaison du numéro de la session à laquelle elle appartient et par son app-id.

Les commandes session-id et unique-id fournissent respectivement le numéro de session et l'identification de l'application active :

```
get-link 'session-id func [ data ] [
    print [ "Numéro de session:" data ]
]
get-link 'unique-id func [ data ] [
    print [ "identification de l'application active:" data ]
]
```

La figure 18-6 illustre l'obtention des informations sur la session active.

Figure 18-6 Informations relatives à la session

Le mot get-link offre un moyen simple d'obtenir des informations diverses sur le client Link. Il présente toutefois le désavantage de bloquer l'exécution du script lors de l'attente d'une réponse. Certaines opérations qui exigent de nombreux traitements sont donc difficilement applicables avec get-link du fait que celles-ci peuvent imposer un temps d'attente trop important.

Pour cette raison, mais seulement pour certaines opérations, vous disposez du mot send-link, qui permet de poursuivre l'exécution du code et éventuellement de réagir à la réception d'une réponse.

Questionner et poursuivre l'exécution avec send-link

Le mot send-link transmet une commande accompagnée de paramètres et confie l'attente de la réponse à un gestionnaire d'événements. L'exécution du script se poursuit une fois la commande transmise.

Si une réponse doit être reçue par le client, vous devez mettre en place un gestionnaire d'événements qui appelle une fonction dédiée à la réception des données. Dans ce cas, l'utilisation de send-link vous permet, par exemple, d'afficher une boîte de dialogue contenant une animation pour faire patienter l'utilisateur.

La syntaxe de send-link comprend les trois paramètres suivants :

- Le nom de la commande.
- Un argument optionnel selon la commande employée et dont la valeur est none si celui-ci n'a aucune signification.
- Le nom du fileset auquel s'applique la commande. Si aucun fileset n'est concerné, ce paramètre est initialisé avec la valeur none.

À la différence de get-link, le mot send-link ne dispose que de trois commandes.

Les commandes utilisables avec send-link

Les commandes exploitables avec send-link sont des actions demandées au client Link. Avec elles, vous pouvez redémarrer le client Link, forcer le maintien de la connexion d'un client Link à un serveur Express, obtenir le contenu d'un fileset ou encore télécharger un fichier présent sur le serveur.

Avec la commande reboot, vous forcez votre client Link à redémarrer en stoppant les différentes reblets actives. À son lancement, les fichiers de configuration sont lus à nouveau. Pour utiliser cette commande, vous ne devez pas spécifier d'argument supplémentaire, et aucun fileset n'est concerné.

Sa syntaxe est la suivante :

```
send-link 'reboot none none
```

La commande keep-alive est transmise avec une valeur booléenne indiquant si le client doit maintenir une connexion avec le serveur :

```
send-link 'keep-alive none true
```

Vous disposez également de la commande get pour déclencher la lecture des fichiers contenus dans un fileset ainsi que de tags le décrivant. Pour utiliser l'une de ces deux options, il vous suffit d'utiliser respectivement app-files et app-reg comme paramètres de la commande.

Pour obtenir, par exemple, la liste des fichiers contenus dans le fileset documents, il vous suffit d'utiliser la syntaxe suivante :

```
send-link 'get 'app-files 'documents
```

Avec la commande download, le client Link peut demander à recevoir un fichier présent sur le serveur dans un fileset non synchronisé, c'est-à-dire ayant un degré de priorité (tag priority) fixé à la valeur none.

Le bloc transmis contient deux paramètres, le nom du fileset contenant le fichier et le chemin d'accès vers ce dernier. L'exemple suivant permet à Link de récupérer le fichier test.txt membre d'un fileset privé de l'administrateur :

```
send-link 'download none [
    documents!admin!!4017014
    %documents/admin/test.txt
]
```

Les commandes get et download présentent une particularité que vous retrouverez dans la suite de l'ouvrage. Elles nécessitent la mise en place d'une gestion d'événements, c'est-à-dire la déclaration d'un événement dont le déclenchement provoque l'abandon de l'opération en cours et l'exécution d'une fonction de traitement.

Gestion des événements

Lorsque vous utilisez le mot send-link afin de transmettre une commande du dialecte d'administration au client, vous devez mettre en place une gestion d'événements destinée à obtenir le résultat du traitement ou à savoir si l'action demandée s'est bien déroulée.

Ce mécanisme est non bloquant puisqu'il permet la poursuite de l'exécution du script ainsi que le lancement simultané de plusieurs commandes gérées par différents événements.

Pour gérer les événements, le dictionnaire de Rebol/Link dispose des mots insert-notify et remove-notify, destinés respectivement à la déclaration d'une fonction responsable du traitement d'un événement particulier et de la suppression de la gestion de l'événement.

Déclarer un événement

Pour déclarer la gestion d'un événement particulier, vous devez utiliser le mot insert-notify. Il permet de préciser quelle fonction doit être appelée lorsqu'un

événement particulier est intercepté par Rebol/Link. Un événement est généré lorsqu'une action, demandée au client ou au serveur, est achevée.

Le mot insert-notify reçoit les trois arguments suivants :

- Le type de l'événement recherché.
- Le fileset pour lequel les événements sont attendus.
- La fonction appelée lorsque l'événement est survenu.

L'événement est intercepté uniquement s'il concerne le fileset indiqué. Il est possible de surveiller l'ensemble des filesets actifs à l'aide de la valeur none.

La fonction appelée reçoit deux paramètres, la nature de l'événement et le bloc de données résultant de l'action initiale. Par convention, il est recommandé d'utiliser event et data pour les noms des paramètres reçus, mais cela n'a rien d'obligatoire.

L'exemple suivant déclare une fonction déclenchée lorsque survient un événement get provoqué par l'exécution de la commande get sur le fileset documents :

```
insert-notify 'get none func [ event data ] [
    print mold data input
]
send-link 'get 'app-files 'documents
```

Vous pouvez déclarer séparément votre fonction afin de rendre votre code puis lisible :

```
on-get: func [ event data ] [
    print mold data input
]
insert-notify 'get none :on-get
send-link 'get 'app-files 'documents
```

Lorsqu'un événement est devenu inutile, vous pouvez le désactiver à l'aide du mot remove-notify.

Supprimer un événement

Sauf pour des cas particuliers, tels que le suivi des erreurs d'exécution sur le serveur Rebol/Express, il est recommandé de désactiver une gestion d'événements lorsque celle-ci n'a plus d'utilité.

Pour cela, vous devez employer le mot remove-notify suivi des deux arguments suivants :

- Le nom de l'événement dont la surveillance n'est plus nécessaire.
- Le nom du fileset auquel est rattachée la gestion de l'événement. Si tous les filesets sont concernés, vous pouvez utiliser la valeur none.

L'exemple suivant initialise une gestion de l'événement get. Lorsqu'il est intercepté, la surveillance de celui-ci est immédiatement supprimée :

```
on-get: func [ event data ] [
    remove-notify 'get none
    print mold data input
]
insert-notify 'get none :on-get
send-link 'get 'app-files 'documents
```

Au même titre que les filesets, la gestion des événements constitue l'un des aspects fondamentaux de Rebol/IOS. Leur compréhension ouvre les portes du développement de véritables applications pour IOS, dépassant le cadre de la simple reblet, afin de mettre en œuvre une relation client-serveur étroite.

Développer une application IOS

Une application IOS est une reblet dont le fonctionnement est tributaire d'opérations réalisées à sa demande sur le serveur Express. L'exécution de l'application est alors répartie entre le poste client et le serveur selon les compétences de chacun.

Ce mode opérationnel permet l'exploitation du mécanisme de synchronisation intégré à Rebol/IOS et la génération de documents pouvant être distribués aux différents membres d'un groupe de travail.

Dans cette optique, vous devez définir un ou plusieurs filesets spécifiques de votre application. La fonction de ceux-ci ne se limite pas à la définition des droits d'accès des utilisateurs ou de l'apparence de l'icône figurant sur le bureau Link. Il s'agit surtout de déterminer un espace de stockage sur le serveur pour les fichiers de données qui seront construits ou consultés par votre applications puis échangés avec les autres utilisateurs.

Pour dialoguer avec le serveur, le client Link utilise le mot send-server afin d'exploiter le dialecte d'administration du serveur Express. Vous disposez de nombreux événements pour surveiller l'exécution des commandes et gérer d'éventuelles erreurs survenant sur le serveur.

Le dialecte d'administration de Rebol/Express

Tout comme Rebol/Link, le serveur Express dispose d'un dialecte d'administration permettant aux reblets de lui demander de réaliser telle ou telle opération. Le mot send-server vous donne accès à de nombreuses fonctionnalités, telles que la gestion des fichiers, la manipulation des filesets ou encore l'administration du serveur. Prenez garde toutefois que Rebol/IOS limite l'emploi de ces opérations aux seuls utilisateurs autorisés selon les droits qui leur ont été attribués lors de la création de leur compte.

Communiquer avec le serveur à l'aide de send-server

Le mot send-server permet à un client Link de transmettre une commande au serveur Express. Celle-ci est accompagnée d'un paramètre contenant none si aucune information complémentaire n'est nécessaire pour l'exécution de la commande. Tout comme pour send-link, l'exécution du code placé à la suite du mot send-server n'est pas suspendue dans l'attente de la réponse du serveur.

Pour que la commande soit utilisable, le client Link émetteur doit obligatoirement être connecté au serveur. Si un résultat est attendu, vous devez mettre en place une gestion d'événements fondée sur le même modèle que celui détaillé précédemment pour la commande send-link. La fonction chargée du traitement de l'événement est déclarée avec insert-notify. La gestion de l'événement peut être désactivée à l'aide du mot remove-notify.

L'exemple suivant utilise les commandes noop et ping. Ces deux commandes sont probablement les plus simples d'emploi. Elles ont la même fonction, qui est de vérifier que le client Link peut communiquer avec le serveur Express. Elles ne nécessitent aucun paramètre additionnel et déclenchent respectivement les événements serve-done et status lorsque le serveur répond.

Pour la commande noop, le serveur Express retourne uniquement au client la valeur 'ok. La commande ping permet d'obtenir l'heure courante sur le serveur Express :

```
on-ping: func [ evt data ] [
    remove-notify 'status none
    print join [ "heure courante: " data ]
    input
]

on-noop: func [ evt data ] [
    remove-notify 'serve-done none
    print mold data
    either data = 'ok [
        print "le serveur est 'ok'"
        insert-notify 'status none :on-ping
        send-server 'ping none
        do-events
    ] [ alert "Impossible de communiquer avec le serveur" ]
]

either connected? [
    insert-notify 'serve-done none :on-noop
    send-server 'noop none
    do-events
] [ alert "Vous devez être connecté au serveur Express" ]
```

Le dialecte d'administration de Rebol/Express dispose de nombreuses et puissantes fonctions, qui permettent d'écrire rapidement des applications tirant parti de l'architecture distribuée de Rebol/IOS.

Envoyer des fichiers

Le dialecte d'administration de Rebol/Express comporte une commande add-file, destinée à transférer un fichier du poste client vers le serveur. Cette commande déclenche l'événement file-downloaded lorsque l'opération est terminée.

La commande add-file utilise un bloc comme paramètre. Celui-ci contient :

- Le nom du fileset auquel sera ajouté le fichier.
- Le chemin d'accès au fichier sur le serveur Express.
- Les données au format binaire compressé contenues dans le fichier.

L'exemple suivant ajoute un fichier nommé test.txt contenant une chaîne de caractères au fileset documents. Les fichiers étant enregistrés dans le répertoire data du serveur Express, le chemin ./documents/test.txt correspond en fait à /data/documents/test.txt dans l'arborescence du serveur Express :

```
REBOL [ type: 'link-app ]

on-add-file: func [ event data ] [
    remove-notify 'file-downloaded none
    print "Transfert terminé"
    input
]
```

```
insert-notify 'file-downloaded none :on-add-file

send-server 'add-file reduce [
    'documents
    %./documents/test.txt
    (compress to-binary "contenu du fichier")
]
do-events
```

Avec la commande add-file, le fichier créé est ajouté à un fileset et est donc synchronisé avec les autres utilisateurs ayant les droits d'accès suffisants. Par contre, il n'apparaît pas sur le bureau de Link puisque aucune icône n'est sélectionnée. Pour rendre ce document visible, vous devez utiliser les commandes destinées à manipuler le contenu de la base de registre IOS.

Manipuler le contenu de la base de registre IOS

Le développement pour Rebol/IOS implique souvent la manipulation des filesets. Le dialecte d'administration de Rebol/Express rend cette opération extrêmement dynamique, puisque vous disposez de trois commandes puissantes afin de créer un fileset, de modifier son contenu ou encore de le supprimer. Toutes les commandes relatives aux filesets déclenchent un événement serve-done lorsque l'opération a pu être réalisée.

La commande new-app permet la construction d'un nouveau fileset. Si ce dernier existe déjà, le fileset présent est remplacé par la nouvelle version. Cette commande nécessite les trois paramètres suivants :

- le nom du fileset ;
- un bloc contenant les tags décrivant le fileset ;
- la liste des fichiers contenus dans le fileset.

L'exemple suivant demande au serveur Express de créer un fileset nommé mon-fileset ne contenant aucun fichier et accessible aux utilisateurs olivier et admin avec des droits étendus :

```
on-new-app: func [ event data ] [
    print mold data
]

tags: [
    users: [ "olivier" "admin" ]
    access: [ properties: rights: delete: change: [ "olivier"
"admin" ] ]
]

insert-notify 'serve-done none :on-new-app
send-server 'new-app reduce [
    'mon-fileset
    tags
    []
]
do-events
```

Avec la commande change-tags, vous pouvez aisément modifier la définition d'un fileset existant. Cette commande a besoin de deux paramètres, le nom du fileset manipulé et un bloc contenant l'ensemble des tags décrivant le fileset.

L'exemple suivant ajoute l'utilisatrice nathalie à la liste de ceux ayant un droit d'accès au fileset mon-fileset. Le script récupère tout d'abord le contenu de la base de registre locale à l'aide la commande get. Le résultat parvient à la fonction on-get, qui extrait le descriptif du fileset mon-fileset. Il suffit alors

d'ajouter le nouvel utilisateur au champ users et de transmettre au serveur la nouvelle définition du fileset à l'aide de la commande change-tags :

```
on-change-tags: func [ event data ] [
    remove-notify 'serve-done none
    print "Utilisateur ajouté"
]

on-get: func [ event data ] [
    remove-notify 'get none
    tags-mon-fileset: do select data 'mon-fileset
    append users "nathalie"
    nouveaux-tags: compose/only [
        users: (users)
        access: (access)
    ]
    insert-notify 'serve-done none :on-change-tags
    send-server 'change-tags reduce [
        'mon-fileset
        nouveaux-tags
    ]
]

insert-notify 'get none :on-get
send-link 'get 'app-reg 'mon-fileset
do-events
```

La commande delete-app vous permet de détruire un fileset. Celle-ci ne reçoit qu'un seul paramètre, qui est le nom du fileset devant être effacé. Pour supprimer le fileset mon-fileset, lancez le script suivant :

```
on-delete-app: func [ event data ] [
    remove-notify 'serve-done none
    print "fileset supprimé"
]

insert-notify 'serve-done none :on-delete-app
send-server 'delete-app 'mon-fileset
do-events
```

À l'aide du dialecte d'administration de Rebol/Express, une reblet peut donc aisément manipuler les filesets de Rebol/IOS, ainsi que ses mécanismes de partage de fichiers et de synchronisation.

Administrer les comptes utilisateur

Le dialecte d'administration de Rebol/Express contient plusieurs commandes destinées à administrer les comptes utilisateur. Grâce à elles, vous pouvez obtenir des informations sur les membres du groupe de travail mais également créer, modifier ou supprimer des utilisateurs et des groupes.

Les commandes user-info et users sont destinées à demander au serveur la liste des utilisateurs. La différence entre ces deux commandes se situe au niveau de la nature des informations obtenues. Avec user-info, votre reblet reçoit le statut de chaque utilisateur (nom du compte utilisateur, localisation, activité, adresse e-mail). Ces informations correspondent à celles affichées dans la reblet who :

```
on-user-info: func [ event data ] [
    print mold data
]

insert-notify 'user-info none :on-user-info
send-server 'user-info none
do-events
```

La figure 18-7 illustre l'obtention des informations sur les comptes utilisateur.

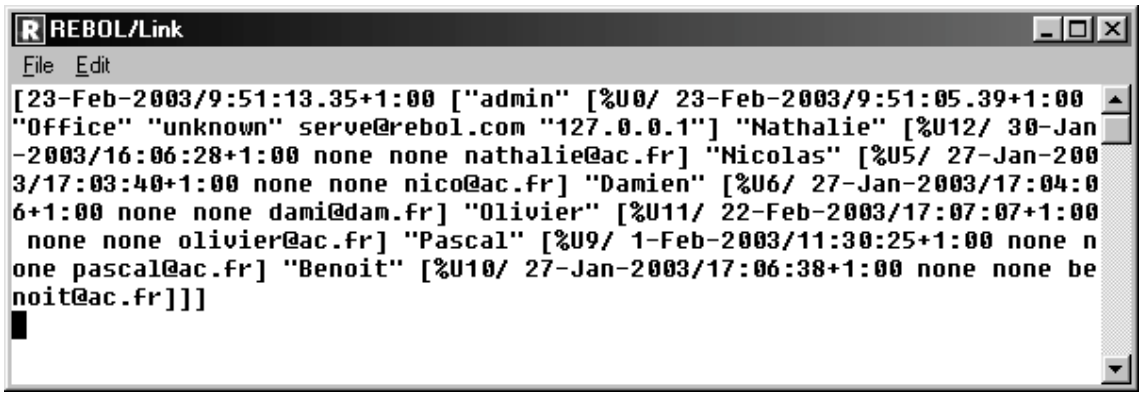


Figure 18-7 Le résultat de la commande user-info

Pour simplifier votre travail, vous avez la possibilité de spécifier la liste des champs qui vous intéressent dans la description du compte de l'utilisateur. Pour les sélectionner, il vous suffit de les passer en argument à l'aide d'un bloc.

L'exemple suivant obtient le nom et l'adresse e-mail de chaque utilisateur :

```
insert-notify 'user-info none :on-user-info
send-server 'user-info [ [ name email ] ]
do-events
```

Le tableau 18-2 récapitule les informations disponibles avec user-info.

Tableau 18-2 Informations disponibles avec user-info

CHAMP	DESCRIPTION
name	Nom du compte utilisateur
Email	Adresse e-mail de l'utilisateur
Time-stamp	Date et heure de la dernière connexion
Home	Nom du répertoire utilisateur sur le serveur Express
Location	Localisation de l'utilisateur
Status	Activité de l'utilisateur
Client-ip	Adresse IP du poste client

À l'aide du paramètre active, vous pouvez savoir quels sont les utilisateurs actuellement connectés au serveur :

```
insert-notify 'user-info none :on-user-info
send-server 'user-info [ active ]
do-events
```

La commande users permet d'obtenir des informations complémentaires sur chaque compte utilisateur. En supplément du compte et de l'adresse e-mail de chaque membre de votre groupe de travail, vous disposez du ou des groupes auxquels appartient l'utilisateur ainsi que de la description de ses droits :

```
on-users: func [ event data ] [
    print mold data
]

insert-notify 'users none :on-users
send-server 'users none
do-events
```

La figure 18-8 illustre l'affichage des informations détaillées disponibles pour chacun des comptes utilisateur.

Figure 18-8 Les droits et groupes de chaque utilisateur

Le dialecte d'administration de Rebol/IOS dispose de nombreuses commandes autorisant la modification des comptes utilisateur. Les différentes opérations provoquent toutes un événement serve-done lorsque le serveur a terminé avec succès le travail demandé.

La commande new-user est destinée à la création d'un nouveau compte utilisateur. Les paramètres transmis à l'aide d'un bloc sont les suivants :

- nom du compte ;
- mot de passe de l'utilisateur ;
- adresse e-mail.

L'exemple suivant est un court script permettant de saisir les informations nécessaires à la création d'un nouveau compte. Une fois les données obtenues, la reblet demande la création du compte au serveur :

```
on-new-user: func [ event data ] [
    print mold data
]

login: ask "Compte:"
password: ask "Mot de passe:"
email: to-email ask "Adresse email:"

insert-notify 'serve-done none :on-new-user
send-server 'new-user reduce [ login password email ]
do-events
```

Le compte est ajouté mais ne dispose d'aucun droit d'accès. Pour modifier ces droits, vous devez utiliser la commande change-user, dont les paramètres sont

l'adresse e-mail de l'utilisateur concerné et un objet contenant les propriétés du compte pour lesquelles vous voulez spécifier de nouvelles valeurs.

Les propriétés admises sont les suivantes :

- name, qui est le nom de l'utilisateur.
- email, qui contient l'adresse de courrier électronique.
- groups, qui est un bloc spécifiant les groupes auxquels appartient l'utilisateur.
- rights, qui est un bloc indiquant les droits de l'utilisateur (post, new-app, user-info, users, instances, operator, user-admin, serve-code, serve-admin).

L'exemple suivant modifie le compte de l'utilisateur ayant l'adresse de courrier électronique manu@perspective.org. Cet utilisateur est affecté au groupe des développeurs, et ses droits sont définis à post et user-info :

```
on-change-user: func [ event data ] [
    print mold data
]

infos-utilisateur: context [
    groups: [ developpeurs ]
    rights: [ post user-info ]
]

insert-notify 'serve-done none :on-change-user
send-server 'change-user reduce [
    manu@perspective.org
    infos-utilisateur
]
do-events
```

Si vous ne voulez modifiez que le mot de passe d'un utilisateur, vous disposez de la commande change-password, dont les paramètres sont l'adresse e-mail de l'utilisateur et le nouveau mot de passe :

```
on-change-password: func [ event data ] [
    print mold data
]

insert-notify 'serve-done none :on-change-password
send-server 'change-password [ manu@perspective.org "gaston" ]
do-events
```

Avec les commandes add-group et remove-group, vous pouvez respectivement ajouter un utilisateur à un groupe ou supprimer l'appartenance d'un utilisateur à un groupe. Ces deux commandes utilisent les mêmes paramètres, à savoir l'adresse e-mail de l'utilisateur et le nom du groupe.

Si un utilisateur est ajouté à un groupe qui n'existe pas, celui-ci est créé dynamiquement par Rebol/Express. L'exemple suivant ajoute l'utilisateur olivier au groupe auteurs. Une fois l'opération réalisée, celle-ci est annulée :

```
on-remove-group: func [ event data ] [
    remove-notify 'serve-done none
    print mold data
]

on-add-group: func [ event data ] [
    remove-notify 'serve-done none
    insert-notify 'serve-done none :on-remove-group
    send-server 'remove-group [ olivier@ac.fr "auteurs" ]
]

insert-notify 'serve-done none :on-add-group
send-server 'add-group [ olivier@ac.fr "auteurs" ]
```

```
do-events
```

Si un utilisateur est devenu inutile et que vous vouliez détruire son compte sur le serveur Express, il vous suffit d'utiliser la commande delete-user, dont le seul paramètre est l'adresse e-mail de l'utilisateur :

```
on-delete-user: func [ event data ] [
    print mold data
]

insert-notify 'serve-done none :on-delete-user
send-server 'delete-user manu@perspective.org
do-events
```

Vous voici au terme de votre découverte des nombreuses fonctions présentes dans le dialecte d'administration de Rebol/Express. Avant de découvrir comment fabriquer vos propres fonctions afin d'étendre les possibilités du serveur, il vous faut connaître les moindres détails des événements système. Ces derniers vous fournissent des informations sur l'état du client Link et du serveur Express.

Capturer les événements système

Jusqu'ici, vous avez mis en œuvre la gestion des événements afin de détecter que l'opération demandée au serveur a été réalisée. Ce faisant, vous avez uniquement utilisé les événements directement liés aux commandes des dialectes d'administration du client Link et du serveur Express.

Le tableau 18-3 recense les principaux événements déclenchés par le client Link et le serveur Express.

Tableau 18-3 Récapitulatif des principaux événements

	FILE-DOWNLOADED	GET	POST-REPLY	SERVE-DONE	STATUS	USERS	USER-INFO
get		✓					
download	✓						
add-file	✓						
new-user				✓			
delete-user				✓			
Change-password				✓			
Change-user				✓			
user-info							✓
users						✓	
add-group				✓			
Remove-group				✓			
new-app				✓			
change-tags				✓			
delete-app				✓			
ping					✓		
noop				✓			

Vous disposez de nombreux autres événements, qui, si vous vous placez dans la logique de Rebol/IOS, peuvent être considérés comme de bas niveau, puisqu'ils surveillent les connexions et déconnexions des utilisateurs, la réception des filesets et des fichiers ou encore les erreurs d'exécution du serveur Express.

Connexion et déconnexion

À l'aide des événements connect et disconnect, une reblet est avertie lorsque l'utilisateur du client Link se connecte au serveur Express (mode "local") ou se déconnecte de celui-ci.

L'exemple suivant définit deux fonctions appelées selon l'état de la connexion :

```
REBOL [ type: 'link-app ]

on-connect: func [ event data ] [
    alert "Vous venez de vous connecter"
]

on-disconnect: func [ event data ] [
    alert "Vous venez de vous déconnecter"
]

insert-notify 'connect none :on-connect
insert-notify 'disconnect none :on-disconnect

view/new layout [
    text "Passez en mode 'local' ou connectez vous"
]
do-events
```

Il est important de savoir que l'état de la connexion n'est pris en compte par la reblet que lorsque celle-ci est active. Pour tester l'exemple précédent, vous pouvez suspendre ou rétablir la connexion de votre client Link en cliquant en bas à droite de son bureau. Ensuite, c'est uniquement en cliquant sur la fenêtre de la reblet que les informations sur la connexion deviennent disponibles.

Transfert des filesets et des fichiers

L'événement fileset-modified est déclenché par la modification d'un fileset par l'utilisateur courant ou par un autre membre du groupe de travail. Lors de l'initialisation du gestionnaire d'événements, vous pouvez utiliser le second paramètre du mot insert-notify afin de spécifier si un fileset particulier est surveillé ou si l'ensemble des filesets est concerné. Dans ce dernier cas, il vous faut utiliser la valeur none.

L'exemple suivant est une reblet chargée de détecter la moindre modification du fileset documents :

```
REBOL [ type: 'link-app ]

insert-notify 'filesets-modified 'documents func [ event data ] [
    print "le fileset 'documents a été modifié"
]
view/new layout [ text "Attente de la modification d'un fileset" ]
do-events
```

Il est possible de détecter le chargement des filesets ou d'un fileset particulier à l'aide de l'événement fileset-downloaded. Le second paramètre de la fonction appelée par le gestionnaire d'événements contient le descriptif du nouveau fileset copié sur le poste client :

```
insert-notify 'fileset-downloaded none func [ event data ] [
    print "Fileset reçu"
    print mold data
]
```

Pour chaque fichier téléchargé sur le poste client, un événement file-downloaded indique que l'opération a été réalisée avec succès :

```
insert-notify 'file-downloaded 'none func [ event data ] [
    print "Fichier récupéré"
]
```

Lorsque l'intégralité du contenu des nouveaux filesets est transférée sur le poste client, un événement all-files-downloaded est généré afin d'indiquer la fin de l'opération :

```
insert-notify 'all-files-downloaded none func [ event data ] [
    print "Fin du transfert des nouveaux fichiers"
]
```

Avec l'événement progress, vous pouvez connaître l'état d'avancement de l'opération consistant à transférer un fichier du serveur vers le client Link. La fonction chargée du traitement de l'événement reçoit un bloc pour le second paramètre. Il s'agit de deux valeurs numériques vous permettant de déterminer l'état d'avancement de la commande download. Si la seconde valeur est supérieure ou égale à la première, le téléchargement est terminé :

```
insert-notify 'progress none func [event data] [
    print [ (mold data/1) (mold data/2) ]
]
```

Cet événement autorise l'utilisation de la barre de progression du VID afin d'afficher graphiquement l'avancement d'un téléchargement. Si votre style progress a pour nom download-status, la formule suivante vous permet de le mettre à jour correctement durant le transfert du fichier :

```
download-status/data: data/2 / max 1 data/1
```

Rebol/IOS dispose ainsi de nombreux événements permettant d'obtenir des informations sur l'activité du client Link. Pour le serveur Express, vous disposez également d'un événement important, puisqu'il s'agit du traitement des erreurs d'exécution.

Détection des erreurs d'exécution

L'événement serve-error est destiné à détecter une erreur d'exécution sur le serveur Express. Lorsque vous utilisez une commande du dialecte d'administration de Rebol/Express et que vous rencontrez un problème, vous pouvez obtenir la cause de l'échec de votre requête.

L'exemple suivant provoque volontairement une erreur en demandant au serveur d'effacer un fileset n'existant pas et se nommant nimportequoi. Le serveur reçoit la commande et génère une erreur, laquelle est interceptée par l'événement serve-error. La fonction on-error est appelée et affiche à l'écran le message d'erreur contenu dans le paramètre event :

```
on-error: func [ event data ] [
    print mold event
    print mold data
]

on-delete-app: func [ event data ] []

insert-notify 'serve-error none :on-error
insert-notify 'serve-done none :on-delete-app
send-server 'delete-app 'nimportequoi

do-events
```

La figure 18-9 illustre le contenu du message obtenu par le client lorsqu'une erreur d'exécution a été détectée sur le serveur.

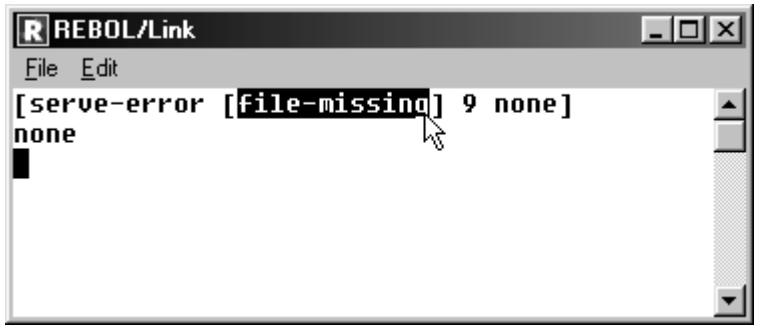


Figure 18-9 Message d'erreur obtenu à l'aide de l'événement serve-error

Ce mécanisme de gestion des erreurs prend toute son importance lorsque vous étendez les fonctionnalités du serveur Express en définissant des méthodes distantes, appelées par les clients et exécutées par le serveur.

Utiliser une méthode post

Rebol/IOS s'appuie sur une répartition intelligente du travail entre le client et le serveur. Il est possible de confier une partie des traitements à Rebol/Express et une autre au client Link de chaque utilisateur.

Une application IOS est donc composée des deux entités suivantes ayant la capacité d'échanger des données :

- une reblet appartenant à un fileset et s'exécutant sur le poste client ;
- une portion de code Rebol appartenant au même fileset que la reblet et s'exécutant sur le serveur.

Le code exécuté par le serveur se nomme une méthode post. Chaque fileset peut contenir une seule de ces méthodes, mais celles-ci peuvent réaliser un nombre quelconque d'opérations selon les requêtes des clients Link. Le code d'une méthode post a accès à une large bibliothèque de fonctions et aux protocoles présents dans le serveur Express.

Dans une application, l'accès aux bases de données peut être centralisé à l'aide d'une méthode post appelée par les clients Link. Il est également possible de faire appel à des applications externes développées avec Rebol/Command ou d'autres langages de programmation.

Mise en place d'une méthode post

Une méthode post est installée sur le serveur lors du transfert d'un fileset. Pour cette raison, le code de la méthode post est inséré dans la définition du fileset de votre application. Le script serveur est rédigé dans un bloc nommé post-func.

Le code de la méthode post étant exécuté dans le même contexte que celui de l'évaluateur Express, vous devez définir les variables spécifiques au contexte du code serveur à l'aide du bloc post-locals. Vous évitez ainsi que les mots définis dans la méthode post n'entrent en conflit avec ceux du dictionnaire de Rebol/Express.

Si le client envoie des données au serveur, ces informations sont disponibles dans un bloc nommé message, dont la longueur dépend du nombre de données

reçues. Le serveur peut retourner des données au client à l'aide du mot return sur le même modèle que celui d'une quelconque fonction.

Pour qu'une reblet puisse dialoguer avec une méthode post du serveur Express, Rebol/Link met à votre disposition une commande nommée post appartenant au dialecte d'administration de Rebol/Express. Cette commande est utilisable à l'aide du mot send-server. Ses arguments sont le nom du fileset contenant la méthode post cible et un nombre quelconque de paramètres, qui sont transmis au serveur, lequel les sauvegarde dans le bloc message vu précédemment.

Lorsque le résultat d'une méthode post est reçu par le client Link, celui-ci déclenche un événement post-reply permettant l'appel d'une fonction dont le troisième élément du deuxième paramètre contient les informations transmises par le serveur Express.

Les différentes étapes d'utilisation d'une méthode post sont illustrées à la figure 18-10.

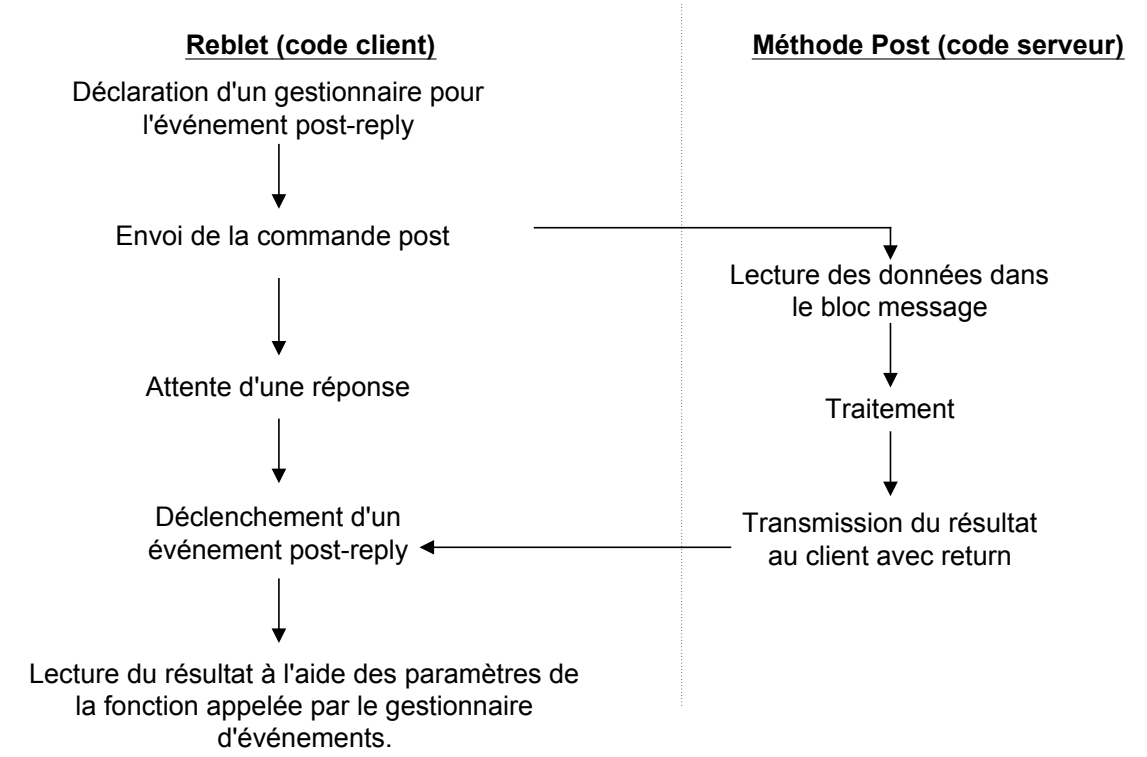


Figure 18-10 Utilisation d'une méthode post

Pour mieux comprendre le fonctionnement et l'utilisation d'une méthode post, vous allez réaliser une petite application IOS, nommée inverse. Celle-ci est composée d'une reblet permettant à l'utilisateur de saisir une chaîne de caractères.

Lorsque l'utilisateur clique sur le bouton Envoyer, le texte est transmis à une méthode post sur le serveur Express. Sa fonction est de renvoyer la chaîne de caractères transmise mais en inversant l'ordre des caractères. Lorsque le résultat est reçu par le client, une boîte de dialogue affiche la réponse de la méthode post.

Le fileset inverse est défini par le fichier install-inverse.r. La reblet est installée dans le classeur Applications de Rebol/Link et est accessible aux utilisateurs admin et olivier. L'intégralité des utilisateurs ayant accès à la reblet peuvent transmettre des données au serveur (post: 'all).

En plus des droits d'accès et de la description de sa structure, le fileset contient une méthode post dans laquelle une donnée transmise par un client Link est affectée à une variable chaine. Le contenu de cette dernière est ensuite inversé avant d'être retourné au client à l'aide du mot return.

Une gestion d'erreur permet d'éviter que la stabilité du serveur Express soit mise en cause par un dysfonctionnement de la méthode post. Si un problème est rencontré, la méthode retourne la valeur none.

La variable chaine est déclarée dans le bloc post-locals afin d'éviter tout conflit avec le dictionnaire de Rebol/Express :

```
REBOL [
    title: "inverse"
    Type: 'link-app
]

secure none

install-fileset [
    fileset: 'inverse
    tags: [
        users: [ "admin" "olivier" ]
        groups: [ ]
        access: [
            properties: rights: delete: change: [
                "admin"
                "olivier"
            ]
            post: 'all
        ]
        icons: [
            [
                name: "inverse"
                item: %apps/inverse/inverse.r
                folder: %apps/
                info: "Inverse l'ordre des caractères"
            ]
        ]
    ]

    files: [
        [%apps/inverse/inverse.r %inverse.r]

    ]
    post-locals: [ chaine ]
    post-func: [
        if error? err: try [
            chaine: copy message/1
            return head reverse chaine
        ] [ return none ]
    ]
]
```

Dans le même répertoire que install-inverse.r, vous pouvez rédiger le script de la reblet inverse.r. L'écran de saisie se compose d'un champ d'édition nommé chaine et d'un bouton. Lorsque l'utilisateur clique sur le bouton, une gestion d'événement post-reply est initialisée pour le fileset 'inverse en déclarant que la fonction affiche-resultat sera appelée à réception de la réponse du serveur.

À l'aide de la commande post, le client appelle la méthode post du fileset 'inverse et lui transmet la valeur saisie dans le champ d'édition. Le script poursuit alors son exécution.

Lorsqu'une réponse est parvenue au client, la fonction affiche-resultat est exécutée. L'événement post-reply est supprimé de la file d'attente des événements, et le résultat est extrait du bloc data. Celui-ci se compose des trois éléments suivants :

- le nom de l'événement ayant déclenché l'appel de la fonction ;
- le nom du fileset concerné par l'événement ;
- les données transmises par le serveur.

Pour afficher le résultat à l'écran, il vous suffit d'utiliser la troisième valeur de ce bloc. Si la valeur est 'none, cela signifie qu'une erreur d'exécution a été rencontrée sur le serveur :

```
REBOL [
    type: 'link-app
]

affiche-resultat: func [ event data ] [
    remove-notify 'post-reply 'inverse
    either data/3 <> 'none [
        alert data/3
    ] [ alert "Erreur sur le serveur" ]
]

view layout [
    chaine: field 200x20
    btn "Envoyer" [
        insert-notify 'post-reply 'inverse :affiche-resultat
        send-server 'post reduce [ 'inverse chaine/text ]
    ]
]
```

Pour tester votre application IOS, il vous suffit de l'installer à l'aide de la séquence de touche **CTRL + L** et de sélectionner le script install-inverse.r. Une fois le transfert terminé, votre reblet apparaît dans l'onglet Applications, comme illustré à la figure 18-11.

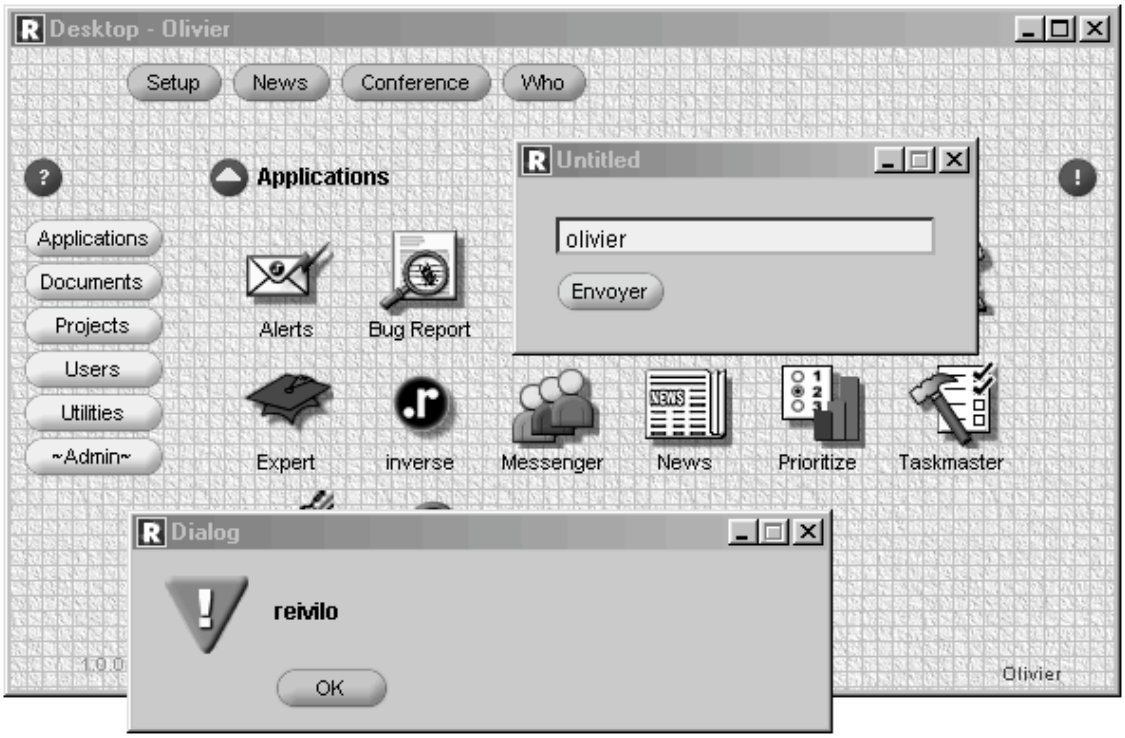


Figure 18-11 La reblet communique avec le serveur Express

Les méthodes post sont simples à mettre en œuvre et surtout d'une grande puissance en ce qu'elles permettent la répartition des traitements entre les clients et le serveur. Rebol/IOS démontre ici ses capacités dans le domaine de l'informatique distribuée et sécurisée. De plus, elle vous permettent d'étendre à l'infini les capacités du serveur Express et de compléter ainsi ses nombreuses fonctions.

Les fonctions spécifiques de Rebol/Express

Rebol/Express dispose d'une large bibliothèque de fonctions autorisant le développement rapide de méthodes post aux possibilités étendues. De nombreux paramètres de fonctionnement de Rebol/IOS peuvent être manipulés à l'aide des scripts exécutés par le serveur.

Parmi ces fonctionnalités, vous allez retrouver certaines des commandes d'administration du serveur Express, ainsi que d'autres, plus spécifiques de l'environnement d'exécution du code serveur.

Trois grands ensembles de fonctions peuvent être mis en évidence puisque le dictionnaire de Rebol/Express contient de nouveaux mots vous permettant de manipuler les filesets, d'administrer les comptes utilisateur et de gérer les fichiers.

Manipuler les filesets

Les méthodes post disposent d'un ensemble de fonctions permettant de gérer complètement les filesets de Rebol/IOS. Vous disposez pratiquement des mêmes possibilités que dans le dialecte d'administration de Rebol/Express. Il n'y a rien

d'étonnant à cela puisque, dans les deux cas, ce sont les mêmes fonctions qui sont utilisées, seule la méthode d'appel changeant.

Pour créer un nouveau fileset ou remplacer la définition d'un fileset existant par une nouvelle description, vous disposez de la fonction new-app. Cette fonction reçoit trois paramètres, le nom du fileset, un bloc contenant la liste des fichiers appartenant aux filesets et un bloc rassemblant les tags décrivant les propriétés du fileset.

Le code suivant construit un fileset nommé test, ne contenant aucun fichier, et dont l'accès est limité aux utilisateurs olivier et admin :

```
new-app 'test [] [
    users: [ "olivier" "admin" ]
    access: [
        properties: rights: delete: change: [ "admin" ]
    ]
]
```

Avec la fonction app-data-name, qui reçoit en paramètre le nom d'un fileset, vous obtenez le nom du fichier contenant ce dernier. Le code suivant affiche dans la console de Rebol/Express le nom du fichier contenant le fileset test :

```
print app-data-name 'test
```

Vous pouvez charger en mémoire la description d'un fileset à l'aide de la fonction load-app, suivie du nom du fileset. Les informations contenues sont alors transformées en un objet afin d'en faciliter l'exploitation. Vous pouvez de la sorte rechercher facilement des informations sur un fileset existant.

L'exemple suivant affiche la description du fileset test dans la console de Rebol/Express :

```
print mold load-app 'test
```

Lorsqu'un fileset est créé, vous pouvez le modifier à l'aide de la fonction change-tags. Celle-ci reçoit deux paramètres, le nom du fileset et un bloc contenant les tags décrivant ses propriétés. L'exemple suivant modifie le fileset test afin d'étendre son accès à l'utilisateur "eric" :

```
change-tags 'test [
    users: [ "olivier" "admin" "eric" ]
    access: [
        properties: rights: delete: change: [ "admin" ]
        post: 'all
    ]
]
```

La fonction add-file permet de créer un nouveau fichier sur le serveur et de l'ajouter à un fileset existant. Le document est alors automatiquement transmis aux utilisateurs ayant accès au fileset contenant le fichier. Cette fonction permet de créer des documents qui seront synchronisés avec les postes client.

L'exemple suivant ajoute un fichier au fileset test. Le document est stocké dans le répertoire /documents de l'arborescence de Rebol/Express. Il porte le nom rapport.txt. Son contenu est au format binaire et contient la chaîne de caractères "rapport" :

```
add-file 'test %/documents/rapport.txt (compress "rapport")
```

Vous avez la possibilité de détruire un fileset à l'aide de la fonction delete-app suivie du nom du fileset. Cette fonction efface toute référence au fileset dans la base de registre IOS. Cette opération est ensuite répétée par les clients Link sur

leur copie locale de la base décrivant les filesets. L'exemple suivant efface le fileset test :

```
delete-app 'test
```

Les filesets de Rebol/IOS sont facilement manipulables à l'aide des fonctions de Rebol/Express. Une méthode post peut créer dynamiquement des filesets ou modifier leur contenu. Ces opérations étant réalisées sur le serveur, leurs effets sont ensuite répercutés par synchronisation sur les clients Link.

Administrer les comptes utilisateur

Rebol/Express dispose de nombreuses fonctions permettant d'administrer les comptes utilisateur dans les méthodes post. Vous pouvez aisément créer, modifier et supprimer des utilisateurs ou encore obtenir des informations sur les différents utilisateurs.

La fonction new-user ajoute un nouvel utilisateur au serveur Express. Sur les quatre arguments disponibles, seuls trois sont utilisés pour indiquer le nom du compte de l'utilisateur, le mot de passe et l'adresse e-mail.

L'exemple suivant crée un utilisateur dont le compte est "pierre" et le mot de passe "carre" :

```
new-user "pierre" "carre" pierre@perspective.org none
```

Lorsqu'un utilisateur existe, vous pouvez modifier les propriétés de son compte à l'aide de la fonction change-user. L'identification de l'utilisateur est assurée par le biais de son adresse e-mail. Le second paramètre est un bloc contenant les propriétés rights et groups. La première définit les droits de l'utilisateur et la seconde la liste des groupes auxquels appartient l'utilisateur.

L'exemple suivant modifie le compte de l'utilisateur "pierre", qui est maintenant rattaché au groupe "test" avec des droits d'administrateur ainsi que la possibilité de transmettre des données au serveur Express :

```
change-user pierre@perspective.org [
    rights: [serve-admin post]
    groups: [ "test" ]
]
```

Avec les fonctions add-user-to-group et remove-user-from-group, un utilisateur peut respectivement être ajouté ou supprimé d'un groupe. Ces deux fonctions reçoivent en paramètre l'identification de l'utilisateur sous la forme de son adresse e-mail et du nom du groupe concerné.

L'exemple suivant ajoute l'utilisateur "pierre" au groupe "gestion" et retire l'utilisateur "patrick" du groupe "test" :

```
add-user-to-group pierre@perspective.org "gestion"
remove-user-from-group patrick@perspective.org "test"
```

Un compte utilisateur peut être détruit au moyen de la fonction delete-user, dont le seul argument est l'adresse e-mail de l'utilisateur :

```
delete-user pierre@perspective.org
```

Rebol/Express met à votre disposition des fonctions permettant d'obtenir des informations sur les comptes utilisateur :

- La fonction serve-total-users fournit le nombre de comptes utilisateur créés sur le serveur.
- La fonction get-users retourne les informations détaillées sur les comptes utilisateur.

- La fonction get-user-info donne les principales informations sur les différents comptes utilisateur, telles que la localisation de l'utilisateur, son adresse IP ou encore son e-mail. Ces informations sont exploitées par la reblet who.
- Le raffinement /active vous permet de ne retenir que les utilisateurs connectés au serveur Express.

L'exemple suivant affiche dans la console de Rebol/Express le nombre d'utilisateurs, ainsi que les informations sur l'ensemble des utilisateurs et la liste des utilisateurs connectés au serveur :

```
print [ "nbr d'utilisateurs: " serve-total-users ]
print mold get-users
print mold get-user-info/active
```

Les méthodes post ont un accès complet aux comptes utilisateur du serveur Express et peuvent ainsi réagir dynamiquement selon leur environnement. Certaines opérations demandées par le client peuvent de la sorte être validées ou refusées selon les droits dont dispose le compte de l'utilisateur.

Gérer les fichiers

Une méthode post peut manipuler les fichiers présents dans le répertoire data de l'arborescence de Rebol/Express. C'est dans ce répertoire que sont stockés les fichiers de travail de Rebol/Express, ainsi que ceux pouvant être synchronisés avec les clients Link. Pour cela, vous disposez d'un ensemble de fonctions capables de créer un fichier, de lire un document ou encore d'obtenir des informations sur un fichier.

Généralement, une application IOS est stockée dans un répertoire contenant les différentes ressources nécessaires à son utilisation (script de la reblet, icône, fichiers synchronisés avec les clients et fichiers de travail uniquement destinés au serveur).

Il est recommandé de prendre la bonne habitude de définir une variable contenant le chemin d'accès au répertoire où se trouve votre application. Cela facilite la maintenance et la lecture de votre code.

Par exemple, pour une application nommée test, présente dans le répertoire apps, vous pouvez définir une variable base de la façon suivante :

```
base: %apps/test/
```

Pour créer un fichier à cet endroit, vous disposez de la fonction write-data, qui utilise deux paramètres, le chemin d'accès au fichier et son contenu sous la forme d'un bloc. Il est très important de retenir que ce fichier n'est pas rattaché à un fileset et n'est donc jamais synchronisé avec les clients Link.

La fonction write-data permet d'enregistrer des fichiers de travail dont l'usage est uniquement réservé aux méthodes post du serveur Express. Si un fichier doit être créé et partagé avec les utilisateurs du fileset auquel il appartient, vous devez obligatoirement utiliser la fonction add-file détaillée précédemment.

L'exemple suivant crée un fichier nommé test.txt contenant la valeur 1 :

```
write-data base/test.txt (mold 1)
```

Pour lire un tel fichier, vous pouvez utiliser les fonctions load-data et read-data, recevant toutes deux comme seul paramètre le chemin d'accès au fichier. Ces deux fonctions ont un comportement légèrement différent.

Avec load-data, le format des données contenues dans le fichier lu est respecté.

La lecture du fichier test.txt renvoie la valeur 1 :

```
print mold load-data base/test.txt
```

La fonction read-data lit un fichier en considérant ce dernier comme une chaîne de caractères. Le format des données n'est donc pas respecté. Le résultat affiché dans la console de Rebol/Express est ici "1" :

```
print mold read-data base/test.txt
```

Grâce aux fonctions size-data? et data-exists?, vous pouvez connaître la taille d'un fichier et savoir si le fichier indiqué existe. Ces deux fonctions reçoivent en paramètre le chemin d'accès au fichier :

```
print size-data? base/test.txt
if data-exists? base/test.txt [ print "fichier trouvé" ]
```

Les méthodes post ont accès à quantité de fonctions et sont extrêmement polyvalentes. Tout est fait pour que leur écriture soit la plus simple et la plus rapide possible.

Accéder aux bases de données

Grâce aux méthodes post, les reblets peuvent accéder aux protocoles intégrés à Rebol/Express. Tout comme les autres versions de Rebol, l'évaluateur du serveur dispose des protocoles réseau fondés sur TCP/IP ainsi que des extensions suivantes de Rebol/Command :

- accès au shell de la machine hôte ;
- fonctions de chiffrement des données ;
- intégration de bibliothèques dynamiques ;
- accès aux bases de données.

Ce dernier aspect est probablement le plus intéressant, car les clients Link ne disposent en standard d'aucun accès aux bases de données. Cela signifie que si une reblet veut accéder à une base de données, elle doit le faire en recourant aux services d'une méthode post, laquelle devient dès lors un fournisseur de données.

Cette solution présente les nombreux avantages suivants sur le plan de la sécurité et du déploiement d'une application :

- Les informations nécessaires à la connexion au SGBD que sont le compte, le mot de passe et le nom de la source de données ne sont pas contenues dans la reblet et sont donc invisibles du client. De plus, ces informations ne transitent jamais sur le réseau.
- Il n'est pas nécessaire de configurer les postes client pour qu'ils accèdent à la base de données. En dehors de Link, aucun logiciel n'est nécessaire. La seule et unique machine communicant avec la base de données est le serveur Express.
- Les échanges de données entre le client Link et le serveur Express étant cryptés, il n'est pas possible pour un utilisateur malveillant d'intercepter le contenu d'une requête SQL ou les données retournées par le serveur à un poste client.

Cette solution présente toutefois un défaut, lié à l'architecture de Rebol/Express. Le serveur n'est pas capable de gérer plusieurs processus simultanément. Cela

signifie qu'il répond aux requêtes des clients en gérant une file d'attente dans laquelle chacun attend son tour.

Pour la gestion des filesets et des fichiers, cela ne pose pas de problème, car Rebol/Express est optimisé pour cela. En revanche, la communication avec une base de données peut être plus problématique si le temps d'exécution des requêtes SQL est important. Lorsqu'une base de données doit être utilisée, vous devez veiller à un usage circonspect de celle-ci.

Avec Rebol/IOS, l'utilisation conjointe des filesets et des structures de données de Rebol permet de ne pas employer obligatoirement un SGBD pour stocker et manipuler des quantités raisonnables de données. L'utilisation d'un SGBD n'est justifiée que si le volume de données est très important ou que vous vouliez intégrer Rebol/IOS dans un système d'information existant. Dans ce cas, reblets, applications bureautiques et applications internet-intranet peuvent alimenter et exploiter conjointement une même base de données.

Rebol/Express pour Linux dispose en natif des protocoles Oracle et MySQL. La version Windows accède aux bases de données disposant d'un pilote ODBC. L'utilisation de ces protocoles est strictement identique à celle présentée au chapitre 15. La seule différence est l'aspect déporté de l'exécution des requêtes SQL puisque le client Link doit demander au serveur Express d'exécuter pour lui une ou plusieurs requêtes SQL.

Pour mettre en place une telle architecture, il existe deux solutions : l'envoi de requêtes SQL à la méthode post ou la rédaction de ces requêtes directement dans le code serveur.

Pour transmettre vos requêtes SQL à une méthode post, il vous faut développer un code générique, dont la fonction est d'exécuter la requête SQL transmise par le client. Cette méthode extrêmement simple à mettre en œuvre permet d'écrire les requêtes SQL dans le script de la reblet. En contrepartie de cette simplicité, le code SQL est visible du client, et l'utilisation des transactions complexe.

L'exemple suivant, extrait d'un fileset test, contient le code d'une méthode post générique dont la fonction est de transmettre une requête SQL à une base de données. La connexion est établie vers la base MySQL db placée sur le serveur db.perspective.org. L'utilisateur se nomme dbuser et son mot de passe est dbpwd :

```
post-locals: [ err con p rep ]
post-func: [
    if error? err: try [
    con: open mysql://dbuser:dbpwd@db.perspective.org/db
        p: first con
        insert p message/1
        rep: copy p
        close p
        close con
        return rep
    ] [
        return false
    ]
]
```

Pour que votre reblet puisse utiliser cette méthode post, il vous suffit d'utiliser le mot send-server. Les paramètres sont le nom du fileset contenant le code d'accès à la base de données et la requête SQL.

Pour recevoir le résultat, vous devez mettre en place une gestion d'événements appelant une fonction chargée du traitement des données lues dans la base de données :

```
insert-notify 'post-reply 'test :reception-data
send-server post reduce [ 'test "SELECT * FROM matable" ]
```

Votre fonction reception-data doit simplement vérifier que les données reçues sont bien un bloc et non la valeur false, indiquant qu'une erreur a été rencontrée lors de la communication du serveur Express avec le SGBD.

Le résultat peut être obtenu par la lecture des données contenues dans le troisième élément du bloc data :

```
reception-data: function [ event data ] [ donnees] [
    remove-notify 'post-reply 'test
    if not block? data [
        request/ok "Les données n'ont pas été lues"
        quit
    ]
    print mold data/3
]
```

Une autre méthode consiste à cacher à l'utilisateur le code SQL en ne le plaçant pas dans le script de la reblet. Les différentes requêtes sont rédigées au sein de la méthode post et appelées à l'aide d'un nom attribué à chacune d'elles. Il suffit alors à la reblet de transmettre l'intitulé de la requête SQL utilisée et les éventuels paramètres à la méthode post.

Cette approche est plus efficace sur le plan de la sécurité puisque l'utilisateur n'a aucun moyen de connaître la structure des tables. Surtout, il devient très simple de mettre en œuvre des transactions, la demande d'une opération à la méthode post —la suppression d'un enregistrement dans plusieurs tables, par exemple — pouvant déclencher l'exécution de plusieurs requêtes.

On peut résumer cette approche comme l'appel de fonctions distantes auxquelles vous passez les paramètres nécessaires à leur exécution.

L'exemple illustré à la figure 18-12 est une petite application imaginaire, destinée à manipuler une base de données contenant des comptes utilisateur. À l'aide de deux champs de saisie, vous pouvez saisir le nom et le prénom de l'utilisateur. Les boutons Utilisateurs et Ajouter permettent respectivement d'obtenir la liste des comptes et d'ajouter un nouvel utilisateur.

Figure 18-12 La reblet permet de manipuler la base de données

L'obtention des différents enregistrements de la base ne nécessite aucun paramètre. La seule valeur transmise à la méthode post est la commande 'utilisateurs.

En revanche, pour ajouter un nouveau compte, le script fait appel à la commande 'ajouter et transmet le nom et le prénom de la personne. Lorsqu'une de ces opérations est terminée, la fonction reponse-sql est appelée par le gestionnaire d'événements :

```
REBOL [
    type: 'link-app
]

reponse-sql: func [ event data ] [
    remove-notify 'post-reply 'sqlpost
    print mold data
]

view/new layout [
    text "Nom:"
    nom: field
    text "Prénom:"
    prenom: field
    btn "Utilisateurs" [
        insert-notify 'post-reply 'sqlpost :reponse-sql
        send-server 'post reduce [ 'sqlpost 'utilisateurs ]
    ]
    btn "Ajouter" [
        insert-notify 'post-reply 'sqlpost :reponse-sql
        send-server 'post reduce [
            'sqlpost 'ajouter
            nom/text prenom/text
        ]       ]
]
do-events
```

Sur le serveur, le fileset contient une méthode post qui ouvre une connexion vers la base de données mabase. À l'aide de la première valeur transmise par le client, la méthode peut déterminer l'opération demandée. Si le client demande la liste des utilisateurs, par exemple, une simple requête select * from users1 est exécutée pour extraire les informations de la table users1. Les informations obtenues sont ensuite sauvegardées dans la variable resultat.

Si l'opération consiste à ajouter un nouveau compte, le script initialise une transaction, car deux tables, users1 et users2, doivent être modifiées simultanément. En cas de réussite, la variable resultat prend la valeur "ok". Si l'opération échoue, c'est la valeur none qui est retournée au client :

```
post-locals: [ requetes db p req resultat ]
post-func: [
if error? err: try [
    db: open odbc://mabase
    switch message/1 [
        utilisateurs [
        p: first db
        insert p "SELECT * FROM users1"
        resultat: copy p
        ]
        ajouter [
        db/locals/auto-commit: false
        update db
        p: first db
        if error? try [
            insert p [
```

```
            {INSERT INTO users1 (nom,prenom) VALUES (?,?)}
        message/2 message/3
            ]
            insert p [
            {INSERT INTO users2 (nom,prenom) VALUES (?,?)}
            message/2 message/3
            ]
            insert db [commit]
            resultat: "ok"
            ] [
                insert db [rollback]
                resultat: none
            ]
        ]
    ]
    close p
    close db
    false
] [ resultat: none ]
return resultat
]
```

Rebol/IOS se révèle extrêmement flexible pour l'exploitation des bases de données. Vous pouvez rapidement et aisément l'intégrer à l'existant d'un système d'information, et ce de différentes façons.

Du fait que les reblets peuvent accéder aux données contenues dans vos bases de données par l'intermédiaire d'une méthode post, vous pouvez générer dynamiquement des documents pour les rattacher à un fileset et les partager entre les membres de votre groupe de travail.

Utiliser des applications externes

Lorsque des opérations gourmandes en temps machine doivent être réalisées, telles que l'extraction d'importants volumes d'information à partir d'une base de données ou le transfert de données sur un réseau, il devient problématique de confier ces travaux aux méthodes post. En effet, celles-ci exécutent chaque tâche selon une file d'attente et non pas simultanément.

Pour résoudre ce problème, Rebol/Express offre la possibilité d'utiliser des applications externes pouvant être écrites en Rebol ou avec un autre langage de programmation.

Cette solution présente les avantages suivants :

- La réactivité du serveur Express n'est pas compromise par les délais qu'impose l'exécution des opérations.
- La stabilité du serveur Express ne peut être mise en défaut par un éventuel problème dans le processus externe.
- Le programmeur a le choix du langage utilisé pour la conception de l'application externe et peut donc éventuellement réutiliser du code existant.

L'exemple qui suit est une application IOS nommée envoimessage, dont l'objectif est d'envoyer un courrier électronique à plusieurs correspondants. Comme illustré à la figure 18-13, l'utilisateur de la reblet saisit un texte dans une zone d'édition et la transmet au serveur Express afin que le message soit expédié à un serveur de messagerie.

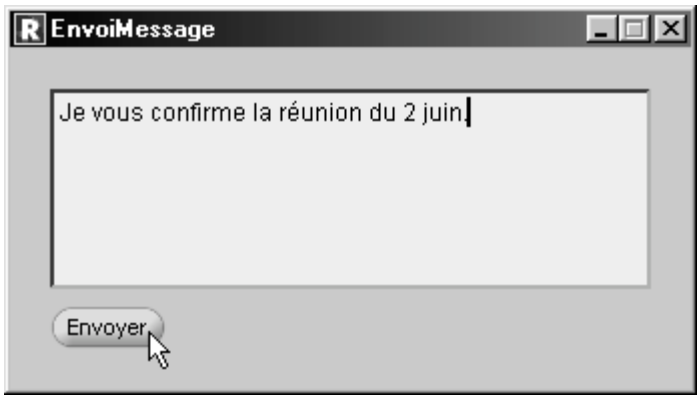


Figure 18-13 L'utilisateur saisit un message à l'aide de la reblet

Étant donné les éventuels délais réseau possibles, l'envoi des courriers électroniques est confié à une application externe, appelée par la méthode post du fileset envoimessage.

La reblet est contenue dans le fichier envoimessage.r. Elle comporte un seul layout, contenant lui-même une zone d'édition pour le message et un bouton permettant de transmettre le texte saisi au serveur à l'aide d'une méthode post :

```
REBOL [
    type: 'link-app
]

view/title layout [
    message: area wrap 300x100 ""
    btn "Envoyer" [
        send-server 'post reduce [ 'envoimessage message/text ]
    ]
] "EnvoiMessage"
```

La description du fileset est confiée au fichier install-envoimessage.r. Celui-ci indique que la reblet est installée dans le classeur Utilities de Rebol/Link.

Seuls les utilisateurs admin et olivier ont la possibilité de l'utiliser. Ce programme transfère deux fichiers sur le serveur Express, le script de la reblet et l'application externe sendmsg.r, que vous rédigez ensuite.

La méthode post ne réalise que deux opérations :

- À l'aide du mot write-data, le texte saisi par l'utilisateur de la reblet est sauvegardé dans le fichier msg.txt au sein du répertoire contenant le fileset de l'application envoimessage.
- La commande call appelle l'évaluateur Rebol/Command en lui passant en paramètre le nom de l'application externe devant être exécutée. Rebol/Command est ici installé dans le répertoire de Rebol/IOS et est accompagné du fichier licence.key, qui est sa clé de licence.

```
REBOL [
    title: "envoimessage"
    Type: 'link-app
]

secure none

install-fileset [
```

```
    fileset: 'envoimessage
    tags: [
        users: [ "admin" "olivier" ]
        groups: [ ]
        access: [
            properties: rights: delete: change: [
                "admin"
                "olivier"
            ]
            post: 'all
        ]
        icons: [
            [
                name: "envoimessage"
                item: %utilities/envoimessage/envoimessage.r
                folder: %utilities/
                info: "Envoi d'un email à une liste de
correspondants"
            ]
        ]
    ]

    files: [
        [%utilities/envoimessage/envoimessage.r %envoimessage.r]
[%utilities/envoimessage/sendmsg.r %sendmsg.r]
    ]
post-locals: []
post-func: [
    if error? try [
        write-data %utilities/envoimessage/msg.txt (mold message/1)
        call "rebolcmd -qws %data/utilities/envoimessage/sendmsg.r"
    ] []
]
]
```

L'application externe sendmsg.r lit le fichier msg.txt afin de prendre connaissance du message saisi par l'utilisateur de la reblet. Le texte est ensuite envoyé aux différents destinataires :

```
REBOL [
    title: "Envoi des messages"
]
message: load %msg.txt
foreach adr [
    pierre@perspective.org   jean@perspective.org
    marie@perspective.org    eric@perspective.org
    manu@perspective.org     maureen@perspective.org
    simon@perspective.org
] [
    send adr message
]
```

Pour créer le fileset sur le serveur Express, vous devez exécuter le fichier install-envoimessage.r à l'aide la séquence de touches **CTRL + L** avec Rebol/Link.

Votre application est maintenant installée, et il vous suffit de cliquer sur l'icône envoimessage dans le classeur Utilities pour saisir et déclencher l'envoi d'un message.

Quel que soit le nombre de destinataires, le serveur Express n'est que très peu sollicité par l'opération, qui est laissée à la charge de l'application externe.

Résumé

Plus qu'un simple produit de groupware, Rebol/IOS est une véritable plate-forme de développement. Avec cette solution, vous pouvez rapidement écrire des

logiciels dont la distribution aux postes client est totalement automatisée et sécurisée.

Fondées sur une architecture distribuée, les applications IOS savent tirer parti de la puissance du poste client et du serveur. Les traitements sont répartis et ne sont plus laissés totalement à la charge du serveur.

Les commandes get-link, send-link et send-server vous donnent accès à deux puissants dialectes d'administration aux fonctionnalités étendues.

Le modèle événementiel utilisé dans Rebol/IOS autorise un développement rapide d'applications parfaitement adaptées aux contraintes des réseaux.

Rebol/IOS dispose ainsi de nombreux atouts pour séduire décideurs et développeurs désireux de mettre en place un système d'information souple, sécurisé et évolutif.