Modules REBOL

REBOL Enhancement Proposal: REP002
Version: 1.0.1
Original Date: 2-Aug-2000
Rev Date: 21-Mar-2005
Author: Carl Sassenrath
(Traduction: Philippe Le Goff, Mai 2006)

NdT : Attention, ce document est une indication sur ce que pourrait être les modules dont parle RT pour REBOL 3.0. Depuis la deuxième version de ce document, revu par Carl, des modifications ont été certainement faites. Une documentation sera réalisée en conséquence par la suite.

[Retour au sommaire]

Présentation

Voici à nouveau la publication de ce document. Cette version, publiée de nouveau, est celle de la proposition initiale des modules en REBOL. Donc, elle peut contenir des informations ou des concepts dépassés. Nous mettrons à jour ce document dès que possible.
Un module REBOL est un contexte indépendant - un espace de noms. Les modules sont similaires aux objets en ce sens qu'ils associent les variables avec leurs valeurs; cependant, les modules étendent le principe des objets de la manière suivante :

Vous ne pouvez accéder aux variables d'un module à moins de les avoir explicitement exportées, sinon vous ne pouvez qu'accéder à la liste de mots d'un module (comme avec le **first** d'un objet).

Similaires à des scripts, les modules incluent un contexte descriptif qui définit le titre du module, la date, la version et d'autres informations qu'il est possible d'exposer (réflectivité).
Vous pouvez créer des modules qui agissent comme un environnement global, protégeant les mots "en dessous d'eux" d'un accès direct.
Vos scripts peuvent être définis comme des modules en précisant cette caractéristique dans l'en-tête du script.

Également, notez que comme les objets, les modules peuvent être encapsulés pour avoir des contextes sous-jacents.

Les modules ne doivent pas être confondus avec les composants qui sont des plugins définissant des caractéristiques internes de REBOL.
D'autre part, un composant peut inclure un ou plusieurs modules.

NdT : La capacité de se faire référence à des valeurs REBOL est liée à une propriété des mots REBOL qu'on appelle le binding.
Seuls les mots liés à un contexte (les mots ayant un contexte) peuvent faire référence à des valeurs REBOL. Définition : les mots n'ayant pas de contextes sont appelés non liés ou libres, Carl utilise le terme de "globals".

[Retour au sommaire]

Le Binding de contexte

L'idée proposée pour un module autorise différents degrés de binding de contexte. Vous pouvez contrôler comment les mots "libres" (non locaux) sont bindés.

*** Strict** : *Lie chaque mot à l'environnement local. Le mot servant à binder devra étendre suffisamment l'environnement local pour manipuler tous les mots. Le module est complètement encapsulé.*

*** Utilisateur (User)** : *Tout mot qui n'aurait pas été défini précédemment dans un contexte sera lié au contexte local. Ceci permet aux utilisateurs d'obtenir les mots pré-éfinis de REBOL mais d'avoir tous les nouveaux mots liés au contexte local. C'est ainsi que la plupart des scripts fonctionneront par défaut.*

*** Explicite** : *Une liste explicite de mots locaux est fournie pour le module. Tous les autres mots disponibles seront liés aux contextes précédents.*

*** Implicite** : *Les mots qui utilisent la notation set-word dans le bloc de plus haut niveau du module seront locaux. Tous les autres seront liés au-dessus. C'est pareil aux variables dans une instance d'objet. C'est le comportement par défaut des modules créés avec la fonction MAKE.*

[Retour au sommaire]

Créer des modules

Les modules peuvent être créés soit avec la fonction MAKE, soit via l'évaluation de scripts qui contiennent le mot "module" dans leur en-tête.

Pour créer un module avec MAKE, vous devez fournir un bloc de spécification d'interface et un bloc pour le corps. Voici la forme générale avec MAKE :

```
new: make module! spec body
```

La valeur retournée par MAKE est le module **new**. C'est une valeur de première classe; elle peut être assignée à une variable, passée à une autre fonction, ou renvoyée comme le résultat d'une fonction.

Voici un exemple de script qui implémente un module :

```
REBOL. [
    title: "Script Module"
    version: 1.0.0
    module: 'script-module ; (the "name" of the module)
    export: [do-it]
]
data: [block of local data]
do-it: does [probe data]
```

Voici un exemple de module dynamiquement créé dans un programme :

```
example: make module! [
    title: "Example module"
    version: 1.0.0
    export: [do-it]
][
    data: [block of local data]
    do-it: does [probe data]
]
```

Spécification d'Interface

Le bloc de spécification d'interface est un en-tête similaire à ceux utilisés dans les scripts. Il contient un titre, une date, la version, le nom de l'auteur, et d'autres champs.
De plus, certains champs sont spécifiques (uniques) aux modules :

export : *Fournit un bloc de mots qui définissent les variables pouvant être accédées de l'extérieur du module.*

import : *Fournit un bloc optionnel de mots importés. Quand ce bloc a été fourni, aucun des mots externes n'est liés au sein du module, à part ceux qui ont été spécifiés. Ceci permet de limiter les accès par les fonctions REBOL au module.*

local : *Spécifie un bloc optionnel qui définit explicitement les variables locales d'un module. Si rien n'est spécifié, alors les variables locales seront définies par les opérations réglées dans le bloc de plus haut niveau (comme avec les objets.)*

options : *un flag spécial utilisé pour contrôler le niveau de binding pour le module. Par exemple, l'attribut LOCAL force les variables disponibles (free) qui ne sont liées à aucun contexte, à être liées dans le module, et non au*

contexte parent. L'attribut USER crée pour le module un contexte de haut niveau (comme un contexte utilisateur).

Dans l'exemple précédent, la fonction *do-it* peut être appelée avec :

```
example/do-it
```

Mais, vous ne pourrez pas accéder à la variable data elle-même. Cette ligne retournera une erreur :

```
example/data
** Script Error: Invalid path value: data
** Where: example/data
```

Des Scripts comme Modules

Pour faciliter les choses, un script peut également définir un module. Ceci est fait en indiquant dans l'en-tête du script la définition du module :

```
REBOL. [
    title: "Script Module"
    version: 1.0.0
    module: 'script-module
    export: [do-it]
]

...
```

Lorsque le script sera évalué, il le sera comme un module. Tous ses mots libres (globaux) seront locaux au module.

Définitions de Variable

Au sein d'un module, les variables sont définies comme locales au module de la même manière que les objets définissent leurs variables d'instance. Vous pouvez écrire :

```
make module! [export [a]] [
    a: b: c: d: e: f: none
    ...
]
```

Vous pouvez explicitement déclarer les variables qui sont locales à un module (mais vous aurez aussi besoin de vous rappelez qu'il faut maintenir cette liste !) :

```
make module! [export [a] local [a b c d e f]] [
    ...
]
```

De telles variables locales seront définies à NONE, lorsque le module sera créé. Bien sûr, si l'option LOCAL est utilisée, alors toutes les variables seront liées localement à la différence de celles qui sont explicitement importées.

[Retour au sommaire]

Hiérarchie des Modules

La nouvelle architecture des modules permet l'amélioration de la structure du contexte pour REBOL et ses scripts.
Les scripts ne seront plus par défaut liés au contexte global (main frame).
Au lieu de cela les scripts seront liés à un contexte utilisateur (USER frame) qui est créé sous le contexte global.

Ceci permet à des scripts entiers et aux fonctions qui y sont définies d'être libérées de la mémoire, si ils ne sont plus nécessaires.

Ceci modifie le processus de binding pour les scripts. Le "binder" regarde en local pour un mot puis consulte la hiérarchie des contextes pour ce mot. Si le mot n'apparaît nulle part dans la hiérarchie, alors le contexte du module est élargi avec le nouveau mot. Effectivement, les variables libres deviennent local au module plutôt que globales.

La hiérarchie des contextes de modules est définie par :

```
ROOT frame (natives, and C code references)
    SYSTEM frame (most of the mezz functions)
        USER frame (user script)
            MODULE frame (sub-scripts or make modules)
```

Lorsque l'option de module USER est utilisée avec MAKE, un contexte de même niveau est créé dans le script. Le nouveau module est placé au niveau du frame USER, plutôt qu'au niveau du niveau MODULE.

Ceci vous permet de créer des scripts qui sont évalués dans des environnements vierges (pristine) (soit, dans le même environnement que votre premier script).

[Retour au sommaire]

Exposition d'un Module

Vous pouvez obtenir l'en-tête d'un module en utilisant la fonction FIRST. Elle renverra l'en-tête tel qu'il a été défini dans le module.
Ceci permet aussi de déterminer quels mots ont été exportés par un module.
Vous pouvez aussi automatiquement générer la documentation d'un module, comme avec les autres en-têtes de scripts REBOL. Example :

```
foreach word get in first module 'export [
    print word
]
```
